VINCENT **DE OLIVEIRA**
CÉDRIC **ESNAULT**

PRÉFACE DE DAVID **ROUSSET**

CSS3

Le design web moderne

DUNOD

Copyright Dunod, 2012
9782100585670

Toutes les marques citées dans cet ouvrage sont des marques déposées par leurs propriétaires respectifs.

Maquette de couverture : barbarycourte.com

Visitez notre site Web : www.dunod.com

Consultez le site Web de cet ouvrage

Le code de la propriété intellectuelle n'autorisant, aux termes des paragraphes 2 et 3 de l'article L122-5, d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective » et, d'autre part, sous réserve du nom de l'auteur et de la source, que « les analyses et les courtes citations justifiées par le caractère critique, polémique, pédagogique, scientifique ou d'information », toute représentation ou reproduction intégrale ou partielle, faite sans consentement de l'auteur ou de ses ayants droit, est illicite (art; L122-4). Toute représentation ou reproduction, par quelque procédé que ce soit, notamment par téléchargement ou sortie imprimante, constituera donc une contrefaçon sanctionnée par les articles L 335-2 et suivants du code de la propriété intellectuelle.

Préface

CSS, un acronyme qui a d'abord un double sens pour moi. Il signifie bien entendu « *Cascading Style Sheets* » que vous allez découvrir en détail dans ce livre. Mais il signifie aussi pour moi la première division dans laquelle j'ai travaillée chez Microsoft et qui veut dire « *Customer Service & Support* ». Mon rôle était alors d'aider nos clients développeurs à revoir et à fixer leur code. C'était en 2000 et l'utilisation de CSS en production n'en était qu'à ses balbutiements. Mais gardez en tête cette idée de revoir un code, nous en reparlerons à la fin de cette préface.

Mais pourquoi est-ce que je commence par vous raconter ma vie ? Pour deux raisons. Tout d'abord, j'ai justement découvert pour la première fois CSS pendant mon activité au support et j'ai donc eu la chance d'en suivre l'évolution. Ses capacités étaient vraiment limitées par rapport à aujourd'hui et peu de gens s'y intéressaient vraiment. Ensuite, pour faire un parallèle à ma propre expérience, je rencontre souvent des personnes qui découvrent la magie des feuilles de styles à travers une expérience de débogage. Cela pose alors souvent problème car au lieu d'essayer de comprendre comment fonctionne la mise en pages contrôlée par des feuilles de styles, beaucoup essaient de bricoler leur CSS en ajoutant/modifiant/enlevant certaines propriétés au petit bonheur la chance. Heureusement, cela ne sera pas votre cas puisque vous avez eu la bonne idée d'acheter ce livre pour mieux comprendre CSS avant de l'utiliser ! Ce sera donc mon premier conseil : *prenez le temps de digérer les subtilités de CSS, elles sont nombreuses mais passionnantes* ! Certains sélecteurs peuvent ainsi parfois vous sembler de prime abord abscons et moins attrayants qu'un magnifique dégradé. Ils sont pourtant très puissants et peuvent rendre des services insoupçonnés en parfois une unique ligne.

Ensuite, au cours de la première décennie, CSS a fini naturellement par exploser. C'est d'ailleurs probablement cette partie des spécifications du W3C qui est le plus à l'origine de notre « culture web » que nous partageons tous. Il nous a ainsi permis un meilleur découpage entre le contenu et la présentation. Les développeurs pouvaient alors commencer à se concentrer sur la production du HTML et du JavaScript pendant que les designers/intégrateurs s'occupaient du look du site *via* CSS. Cette collaboration était normalement vouée à un avenir radieux.

Cependant, en parallèle de cela, il s'est développé un art à part à entière rendant célèbre certains intégrateurs : être compatible avec tous les navigateurs. Rivalisant alors de trésors d'ingéniosité, certains « *hacks* » sont devenus monnaie courante dans la production de CSS. Associé à cela s'est développé un autre problème que vous avez sûrement rencontré : cette volonté à tout prix de faire du web un media ressemblant aux autres. On voulait alors que le site s'affiche de la même manière (au pixel près ou « *pixel perfect* ») sur tous les navigateurs et toutes les machines de la planète ! Tout cela en partant d'une planche produite sous Photoshop. Nous savons aujourd'hui que le problème avait alors été pris à l'envers. Le web doit vraiment être vu comme un media à part entière. Ce sera donc mon deuxième conseil : *ne faites pas un site comme on fait un magazine*.

Heureusement, depuis la fin des années 2000, les choses ont évolué pour beaucoup dans le bon sens. Il y a plusieurs raisons à cela. Tout d'abord, nous avons eu de nouvelles armes côté CSS3 permettant d'envisager de nouvelles approches. Le design adaptatif étant l'une des plus célèbres en ce moment. En effet, l'ergonomie et le design doivent être différents entre un téléphone, une tablette et un PC.

Techniquement parlant, vous allez alors utiliser des « *Media Queries* » ou des contrôles flexibles notamment abordés dans ce livre.

Ensuite, l'approche sur le support multi-navigateurs a changé, les mentalités ont bougé et les techniques se sont améliorées. Sont alors nées les approches dîtes de dégradation gracieuse ou d'amélioration progressive. Les sites ont alors cessé de se limiter aux capacités du plus vieux navigateur toujours en circulation et la culture du « *pixel perfect* » a éclaté au fur et à mesure. En fonction du navigateur ou du périphérique que l'on utilise, on peut désormais avoir une expérience différente sur le même site. CSS et la manière dont il fonctionne intrinsèquement y est pour beaucoup.

Pour finir sur les raisons de cette bonne évolution, il y a bien évidemment le support de ces spécifications par l'ensemble des navigateurs et la concurrence effrénée entre eux. La qualité d'implémentation évoluant de versions en versions, les parts de marché des navigateurs modernes ne cessant d'augmenter, certains cauchemars de l'intégrateur ne seront plus bientôt que de l'histoire ancienne. D'ailleurs dans cette guerre perpétuelle des navigateurs, CSS a toujours joué un rôle primordial et fut toujours au centre des intérêts de ceux qui faisaient le web.

Dans cette deuxième décennie, une nouvelle page de l'histoire du web est en train de se tourner. CSS3 est désormais plus prisé que jamais car c'est le compagnon indispensable à la superstar HTML5. C'est CSS3 qui fera que votre site ou votre application web sera attrayante ou non.

Or les attentes des utilisateurs vis-à-vis des interfaces ont nettement évolué ces dernières années suite aux succès des smartphones et tablettes. Ils veulent des interfaces novatrices, attirantes et efficaces. CSS3 leur proposera pour cela toute une palette d'effets graphiques à base d'ombre, de dégradés, de bords non rectangulaires, de transformations 2D voire même en 3D !

L'avènement du *touch* bouscule aussi la manière de concevoir une interface. Il faut penser aux gros doigts de l'utilisateur mais il faut aussi donner l'impression de fluidité et de réactivité permanente. Si vous touchez l'écran et qu'il ne se passe pas immédiatement quelque chose en retour, vous avez la désagréable sensation d'une expérience dégradée. Pour éviter cela, CSS3 propose la possibilité de gérer des animations sur les éléments *via* des transitions ou des étapes contrôlées d'animations. La bonne nouvelle ? Ces nouvelles animations ne sont désormais plus à la charge du développeur qui les faisaient avant *via* du JavaScript et dont ce n'était pas vraiment le métier. L'autre bonne nouvelle : c'est pris en charge par le processeur graphique du périphérique (GPU) pour des performances souvent meilleures que leurs équivalences JavaScript. Une nouvelle génération d'applications web est donc sur le point de naître grâce à toutes ces nouvelles spécifications.

CSS est donc plus que jamais au centre de toutes les attentions. De notre côté, nous lui offrons une place de choix dans la conception des applications HTML5 Metro pour Windows 8 mais également en améliorant drastiquement son support dans IE10. Par ailleurs, que ce soit chez Adobe ou chez Microsoft, l'outillage arrive également pour augmenter la productivité et par conséquence la rentabilité des investissements fait sur ces technologies.

Mais CSS est également en mouvement permanent car une grande partie de ses spécifications ne sont pas encore finalisées. Comme son usage est désormais stratégique, l'ensemble des éditeurs de navigateurs ne cessent également d'innover en proposant régulièrement de nouvelles fonctionnalités. Pour éviter de freiner l'innovation, le W3C eut alors l'idée de laisser un éditeur implémenter une version expérimentale d'une nouvelle technologie *via* l'utilisation de préfixes (-webkit, -moz, -ms ou -o). C'est une bonne idée mais cela déporte la responsabilité vers vous lecteur et futur utilisateur de CSS.

Mais de quelle responsabilité parle-t-il me direz-vous ? Pouvoir commencer à goûter au futur est

plutôt une bonne nouvelle, non ? Oui ! Mais en utilisant une spécification non finie, charge alors à vous de bien faire attention à plusieurs choses.

Tout d'abord, choisissez avec précaution la fonctionnalité en question que vous allez utiliser pour éviter de rendre l'usage de votre site exclusif à un navigateur unique. L'absence de support de cette fonctionnalité ne doit pas empêcher d'autres navigateurs d'afficher votre contenu (*via* par exemple une technique de dégradation gracieuse que vous verrez plus loin). C'est particulièrement important si vous ne souhaitez pas retomber dans le même genre de problème de culture unique que nous avons connu avec IE6 au début des années 2000.

Par ailleurs, si vous utilisez un préfixe en production, il faudra passer du temps à maintenir cette partie pour tenir compte de l'éventuelle évolution de la spécification et surtout de son support par de nouveaux navigateurs. Si vous ne mettez pas à jour votre site, les nouveaux navigateurs ne pourront pas en bénéficier correctement et par voie de fait vos propres utilisateurs seront touchés.

Ma conclusion et mon dernier conseil s'appuieront donc sur une citation connue des cinéphiles : « *un grand pouvoir implique de grandes responsabilités* ». Grâce à votre maîtrise de CSS, vous allez être au cœur de la future génération d'interface utilisateur pour les smartphones, tablettes et PC. Votre rôle n'aura jamais été aussi important. Mais si vous voulez garder un web ouvert, n'oubliez jamais les autres navigateurs.

David Rousset
Developer Evangelist chez Microsoft France

Avant-propos

Vous venez d'acquérir cet ouvrage dont l'objectif est de vous faire découvrir les apports de CSS3 en matière de design web moderne. Au fil de la lecture, nous vous présenterons les nouveautés de ce langage à l'aide d'exemples concrets qui vous permettront d'assimiler rapidement toutes ces notions.

Vous avez sûrement déjà entendu ou lu ce genre d'affirmation : « *CSS3 c'est génial, CSS3 c'est trop tôt, CSS3 ça ne marche pas partout* », ou tout simplement « *pourquoi a-t-on besoin de CSS3* ? » Mais savez-vous réellement ce qu'il en est ?

Ce livre tente de répondre aux questions que vous vous êtes posées en vous présentant les fonctionnalités du langage, leurs cas pratiques d'utilisations, ainsi que les méthodes à mettre en place pour pouvoir utiliser CSS3 dès aujourd'hui.

Pour cela, nous vous présenterons successivement :

- les spécificités du langage ;
- les avancées graphiques et typographiques ;
- les positionnements modernes ;
- l'adaptation aux différents médias ;
- l'amélioration de l'expérience utilisateur avec les animations et les transformations ;
- des outils utiles ;
- ce que l'on peut attendre de l'avenir.

Bonne lecture.

À qui s'adresse ce livre ?

Ce livre s'adresse à toutes les personnes, travaillant dans le web ou non, désireuses d'apprendre les nouveautés présentes au sein de CSS version 3. Il est conseillé d'avoir une base sur HTML, et surtout sur CSS version 1 et 2.

Les concepteurs web ayant déjà une première approche de CSS3 trouveront dans ce livre des cas précis d'utilisation, ainsi que des méthodes à mettre en place afin d'en profiter dès aujourd'hui sur la majorité des sites web.

Présentation du site support

Afin de vous offrir une expérience de lecture hors du commun (et nous ne pesons pas nos mots), nous avons décidé de mettre à votre disposition un site support sur lequel la majorité des démonstrations présentes dans ce livre prennent vie :

http://livre-css3.fr/bugdroid/

Plus sérieusement, nous ne pouvions pas faire un livre sur CSS3 sans combiner nos explications à des exemples en situation réelle. Tout simplement, pour en comprendre le fonctionnement, les avantages, mais également pour découvrir les contraintes. Les fonctionnalités nouvelles de CSS sont bien entendu utilisables dès aujourd'hui, mais pour certaines, des solutions alternatives doivent êtres mises en place pour contourner les manques de compatibilité. C'est à ces problématiques que le site support

tente de répondre.

Les données nécessaires à la bonne réalisation des exemples sont téléchargeables depuis le site support.

Le site support : http://livre-css3.fr/bugdroid/

Ce site a été créé par nos soins, dans le seul but d'aider à la compréhension. Le contenu est entièrement libre de droits. Le personnage nommé *Bugdroid*, qui nous sert de support, est le petit robot vert utilisé par Google pour présenter Android. Ce personnage est sous licence « creative commons by (3.0) » et peut donc être utilisé librement.

Nous espérons que ce livre et le site de démonstration, vous apportent les clés de la réussite dans votre apprentissage de CSS, et plus principalement de CSS version 3.

Les auteurs

- Cédric vu par Vincent
  Breton pur souche, adepte de kitesurf, pratiquant la course d'orientation, Cédric a écrit ses premières lignes de code à l'âge de 9 ans. Après avoir créé ses premiers jeux et images 3D à la fin des années 1990, il a débuté le développement web en 2001 avec comme ligne de conduite : le respect les standards et l'accessibilité. C'est d'ailleurs ces bonnes pratiques qu'il transmet aux élèves de l'ENSG (École nationale des sciences géographique), destinés à l'IGN, depuis 2005. Touche à tout et extrêmement doué, il code aussi bien en PHP, JavaScript, C, C++, Java, etc. Enfin, partisan de l'open-source, c'est tout naturellement qu'il aime critiquer la marque à la pomme (mais c'est de bonne guerre).
- Vincent vu par Cédric
  J'ai connu Vincent quand il est arrivé à l'ENSG pour enseigner l'infographie, l'animal maniait déjà Photoshop comme un pro, mais il faisait les gros yeux quand je lui parlais de W3C, de XHTML et autre PHP et linuxerie. Après avoir exploré TOUS les templates du CSS ZEN GARDEN, il s'est vite rendu compte que le web était un formidable outil pour exprimer son talent esthétique. Il a alors décortiqué et analysé chaque portion de règle CSS afin d'en tirer le maximum dans des designs toujours plus resplendissants (CSS3create.com). De mon côté, je restais sur mon border : 2 px ridge orange ; mais il en faut pour tous les goûts. Nous avons ensuite participé à un petit concours ensemble (wimt.fr) où nous avons bien exploité nos

complémentarités, une complicité créative était née...

Remerciements

Nous remercions tout d'abord Dunod qui nous permet d'éditer ce livre ainsi que Jonathan Verrecchia (co-auteur du livre HTML5 aux mêmes éditions) qui nous a initialement offert cette opportunité, David Rousset qui nous fait l'honneur de signer la préface, ainsi que tous nos collègues de l'ENSG qui supportent nos CSSries à longueur de journées.

Vincent : « *Je remercie plus particulièrement Cédric qui m'a transmis la passion du web ainsi que Philippe qui a permis mon arrivée à l'ENSG. Je pense ensuite à Renaud et à ses encouragements. Enfin, je remercie ma douce et tendre Auriane pour sa patience, son soutien et sa bonne humeur toujours remotivante*. »

Cédric : « *Je tiens surtout à remercier Vincent qui m'a proposé de coécrire ce livre et qui m'a permis de participer à cet extraordinaire exercice qu'est la rédaction d'un ouvrage. Bien sûr tout ceci n'aurait pas été réalisable sans la patience et le soutient de ma formidable compagne Lætitia, je tiens donc à la remercier tout particulièrement ainsi que mes deux adorables enfants*. »

Enfin, nous remercions ensemble toute la communauté du web, du W3C à Mr Foo qui, derrière sa question idiote nous fait réaliser que rien n'est jamais acquis. Le web est une éternelle remise en question et sans le savoir de chacun, rien ne pourrait avancer. Merci à tous ceux qui partagent leurs connaissances.

État de l'art

CSS3 n'existe pas ! Enfin, ce n'est pas tout à fait vrai. En fait, CSS3 est une évolution logique du langage CSS, le langage des feuilles de styles. Il ne doit pas être vu comme un langage à part entière, mais comme la suite ou le prolongement de la recommandation actuelle CSS2.1. Chaque nouveauté présente dans la spécification n'est qu'une étude de cas, une tentative de modification, de simplification, un ajout de fonctionnalité par rapport à CSS2.1.

Nous devrions alors parler de CSS, tout court. C'est vrai. Pourtant, pour différencier les fonctionnalités nouvelles du langage, nous parlons de CSS3. Mais savez-vous réellement ce qu'est CSS et qui s'occupe de le normaliser ?

1.1. Les standards web

Un standard est un ensemble de spécifications qui définissent la méthode d'utilisation des technologies web. Ce standard est une recommandation qui doit être respectée de tous, navigateurs et concepteurs web, afin d'éviter les différences d'interprétation pour un code identique. De plus, ce standard doit permettre l'accessibilité des données.

Le World Wide Web Consortium (W3C) (figure 1.1) est l'organisme de normalisation dont le rôle est de créer, de développer, de maintenir et de promouvoir les standards du web, en travaillant notamment sur les langages tels que HTML (*HyperText Markup Language*), XHTML (*eXtensible HyperText Markup Language*), CSS (*Cascading Style Sheet*), XML (*eXtended Markup Language*), SVG (*Scalable Vector Graphics*), etc. Le W3C est aujourd'hui composé en grande partie par les éditeurs de navigateurs, que sont Microsoft, Mozilla, Opéra Software, Google ou encore Apple Inc, ainsi que de plusieurs experts. Ils s'accordent sur les développements présents et futurs des standards web.

Fig. 1.1 Logo du W3C

Ce qui nous intéresse ici c'est CSS, le langage des feuilles styles. Le W3C définit le standard ainsi : CSS est le langage qui permet aux concepteurs de pages web de modifier le rendu d'un document structuré, tels les documents HTML. En séparant le contenu de sa mise en forme, CSS simplifie la création et la maintenance des documents web. C'est le principe de la séparation du fond et de la forme.

Pour mettre en place ce principe pour nos pages web, nous disposons de deux langages distincts : le HTML et le CSS. Le HTML permet de décrire les données, c'est le fond. Il n'a pas été créé pour définir le rendu visuel des documents web, mais pour définir la structure du document et organiser le contenu des informations qu'il contient. CSS modifie l'aspect visuel de ces données, c'est la forme. HTML et CSS sont donc indissociables (figure 1.2).

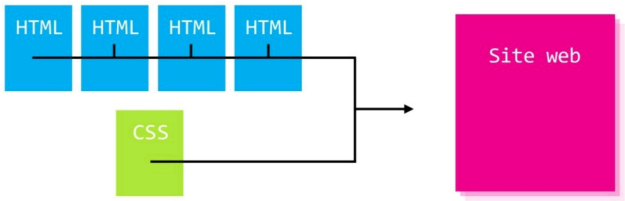


Fig. 1.2 Le principe de la séparation fond/forme

L'intérêt de cette séparation est multiple, elle permet :

- de pouvoir disposer d'une feuille de styles unique, applicable à des documents multiples ;
- de pouvoir disposer de feuilles de styles alternatives, en fonction du choix des concepteurs ou des utilisateurs ou en fonction des médias ;
- d'avoir une meilleure maintenabilité d'un site web (les mises à jour sont plus simples et plus rapides) ;
- d'avoir une plus grande flexibilité (une seule modification du CSS modifie l'ensemble des pages d'un site web).

Mais ce n'est pas tout, CSS permet également l'accessibilité des données en supportant les feuilles de styles spécifiques par média. En effet, les médias alternatifs ne doivent pas être oubliés car la présentation d'une page HTML *via* CSS ne se limite pas à sa version écran classique. Plusieurs types de médias sont connectés au web et donc potentiellement clients de nos pages HTML. Cela comprend les terminaux mobiles bien sûr (téléphones et tablettes), mais également les téléviseurs, les médias embarqués dans l'automobile ou les lecteurs d'écrans qui permettent par exemple aux internautes malvoyants d'accéder à l'information. C'est la base de l'accessibilité du web. CSS peut nous aider à rendre notre contenu accessible pour tout et pour tous.

Cette séparation, bien qu'évidente aujourd'hui, n'a pas été simple à mettre en place. Pour le comprendre, faisons un saut d'une vingtaine d'années en arrière.

1.2. L'histoire du web

Le web tel qu'il a été imaginé par Tim Berners-Lee (qui préside aujourd'hui le W3C), aux alentours de 1990, s'articule autour d'une architecture dite client-serveur. Dans le cas d'une navigation classique, le client c'est nous ou plutôt notre navigateur. Le serveur, c'est un programme qui répond à nos requêtes grâce au protocole HTTP (*HyperText Transfer Protocol*). La notion d'hypertexte est aussi associée à un acronyme que nous connaissons bien, le HTML. Ces deux « inventions » associées au concept d'URL qui complète la notion d'hypertexte, sont les bases de la création du **World Wide Web**.

CSS n'était pas présent à cette étape du web. Pour autant, le HTML n'avait pas vocation à prendre en charge la présentation des données. Les premiers navigateurs proposaient alors déjà une implémentation de type « feuille de styles utilisateur » pour modifier l'aspect des pages HTML brutes.

Bien que CSS soit proposé au W3C dès 1994, il s'impose tardivement dans la bataille de la présentation des documents web et cela pour plusieurs raisons. Tout d'abord, le W3C n'avait pas la reconnaissance qu'il a aujourd'hui. Mais surtout les deux navigateurs de l'époque, qu'étaient Netscape Navigator et Internet Explorer (figure 1.3) – tentant chacun de récupérer les parts de marchés de l'autre – avaient d'autres objectifs. Sous la pression des développeurs web, les deux éditeurs ont

alors choisi de créer des balises et attributs HTML de présentation non conformes W3C, sans commencer à réellement supporter CSS. En effet, c'est à cette période que les balises font, center, b, etc. ont vu le jour. Ces balises sortent du champ d'application de HTML puisque le principe de ces balises est de mettre en forme. De plus, le détournement des balises table, tr et td pour créer la structure d'un site web a retardé le support et l'apprentissage de CSS par les concepteurs web.

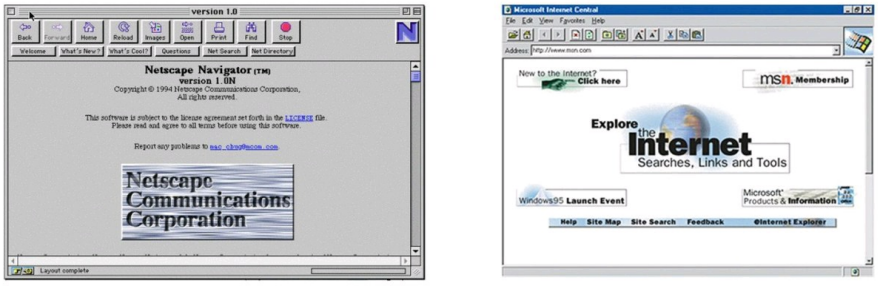


Fig. 1.3 Netscape Navigator 1.0 (1994) et Internet Explorer 1.0 (1995)

En 2001, Internet Explorer sort la version 6 de son navigateur et à cette époque, plus de 90 % du trafic internet est généré depuis ce navigateur (en France). Malheureusement, le support de CSS n'est pas au rendez-vous et la majorité des développeurs web continuent à utiliser les « mauvaises pratiques » du web.

Dans les années qui suivent, de nouveaux navigateurs font leur apparition. C'est le cas de Firefox, d'Opéra, de Safari et de Chrome, pour ne citer que les principaux. Ces navigateurs tentent alors de corriger les imperfections d'Internet Explorer et c'est ce qu'ils arrivent progressivement à faire. En 2004, la machine s'inverse et Internet Explorer commence alors à perdre des parts de marchés. Non pas à cause de sa mauvaise prise en charge des standards (les internautes n'ont que faire des standards) mais à cause de ses piètres performances par rapport à la concurrence. Petit à petit, IE passe du stade de leader du marché au stade d'outsider, sa stratégie est alors bousculée. En 2006, cinq ans après IE6, la version 7 du navigateur sort, sans grand résultat et en 2009 c'est au tour d'IE8. Là encore, le respect des standards est encore à prouver. En 2011, la version 9 rattrape son retard.

Aujourd'hui, le standard CSS2.1 est largement supporté dans tous les navigateurs. Quelques bugs doivent encore être pris en charge pour les versions d'IE7 et 8. CSS3 est, quant à lui, également de plus en plus reconnu, mais c'est surtout la mentalité des développeurs web qui a changé. Aujourd'hui, plus que jamais, le respect des standards s'impose !

1.3. Évolution du standard CSS

La première recommandation CSS1 voit le jour en 1996 (figure 1.4) et définit la majorité des styles de bases que l'on connaît aujourd'hui : les sélecteurs et certaines pseudo-classes et pseudo-éléments, les propriétés de boîtes, de tailles, de marges, de polices, de textes, de couleurs, etc.

Très peu de temps après, en 1998, CSS passe en version 2. Cette version ajoute énormément de fonctionnalités par rapport à la précédente. Elle propose par exemple la création de pseudo-classes dynamiques comme :hover ou :focus, des pseudo-éléments :after et :before, des polices téléchargeables avec @font-face, la gestion des médias alternatifs avec @media, l'ajout d'ombre sur les textes avec text-shadow, etc. Cependant, l'évolution a été trop rapide et trop de parties sont sous-spécifiées voire trop technologiques pour l'époque. Cette version a donc été stoppée et remise à l'étude. La version « corrigée » de CSS2, dont le nom est CSS2.1, est arrivée à l'état de recommandation en juin 2011,

soit treize ans plus tard (figure 1.4). Cette version est, de plus, allégée de plusieurs propositions faites dans CSS2. Ces propositions sont déplacées pour être étudiée dans la future version de CSS.

Fig. 1.4 Recommandation CSS1 et CSS2.1 sur le site du W3C

C'est pourquoi depuis 1999, la spécification CSS3 commence d'être définie parallèlement. Cette nouvelle spécification utilise un modèle de conception différent, basé sur des modules, alors que les versions précédentes sont de longs documents, organisés en chapitres. En 2011, trois modules de niveau 3 (CSS3) ont atteint le statut de recommandation et environ une trentaine de modules sont à l'étude.

La future version, CSS4, est déjà en réflexion pour certains modules, comme le module des *Sélecteurs* ou encore celui des *Bordures et Arrière-plans*.

Mise en garde : CSS3 est un standard web non finalisé et en constante évolution. C'est pourquoi nous vous invitons à une attention toute particulière, ce livre présentant un état des lieux de CSS en mars 2012. Depuis cette date, les spécifications ont évoluées et nous vous conseillons d'en vérifier la conformité.

1.3.1. Niveau de validation

Chaque spécification du W3C suit un processus de validation par étapes et ce jusqu'à sa recommandation qui équivaut à un standard que chaque navigateur doit alors respecter. CSS n'échappe pas à ce principe. Voici la liste des différentes étapes par lesquelles un module CSS passe :

Editor's Draft (ED) – Brouillon éditeur : c'est l'ébauche d'une proposition, faite généralement par les éditeurs de navigateurs. Rien n'est officialisé à ce stade.

Working Draft (WD) – Brouillon de travail : c'est la première étape de la standardisation officialisée par le W3C. C'est un document qui est souvent incomplet et soumis à changements. Le W3C et le public sont invités à se prononcer sur ces propositions.

Last Call (LC) – Dernier appel : à cette étape, le groupe de travail estime que la spécification a atteint une certaine maturité, et que tous les problèmes découverts lors de la phase de WD ont été traités. Les derniers retours sont attendus.

Candidate Recommandation (CR) – Candidat à la recommandation : à ce stade, le groupe de travail estime que l'implémentation par les navigateurs est nécessaire et sans risques. Le W3C encourage une utilisation régulière de la spécification depuis cet état.

Proposed Recommandation (PR) – Proposée à la recommandation : pour atteindre ce niveau, la spécification doit avoir été implémentée et de nombreux tests doivent avoir été mis en place, afin d'être parfaitement sur de la bonne interopérabilité avec l'existant. Si ce stade est validé, la spécification passe en recommandation.

Recommandation (REC) – Recommandation : c'est l'étape ultime. Une recommandation définit les nouveaux standards du web. Plus aucun changement ne peut avoir lieu.

Il n'y a pas de durée représentative concernant le maintien d'une spécification à un stade précis. Tout dépend de l'engouement de la communauté web pour une fonctionnalité proposée, de la complexité à normaliser ou non une propriété, de l'évolution des autres modules, etc.

Dès qu'un module est à l'étude, la communauté web pousse alors les éditeurs de navigateurs à implémenter certaines fonctionnalités. Cependant, la spécification étant encore soumise à changements, les navigateurs ne peuvent proposer que des versions « bêta » des fonctionnalités. Pour ce faire, des versions préfixées de propositions sont créées. Nous reviendrons sur cet aspect un peu plus loin.

1.3.2. Les modules CSS3

Contrairement à CSS1 et CSS2.1, l'organisation sous forme de modules de CSS3 permet une évolution progressive de la recommandation, un module pouvant être finalisé avant les autres. Plusieurs avantages à cela :

- une progression plus rapide de la spécification, au moins en partie ;
- le travail de normalisation est facilité ;
- le contenu d'un module peut être référencé dans un autre (ou plusieurs) ;
- de nouveaux modules peuvent êtres proposés plus facilement.

Ce découpage permet à certains modules d'atteindre le stade de recommandation, sans pour autant attendre que la spécification entière l'ait atteint. C'est pour cela que nous ne pouvons plus parler de CSS3 en globalité mais bien par modules, chaque module ayant son propre niveau d'avancement. Il est même probable que certains modules CSS4 soient finalisés avant la totalité des modules CSS3.

Voici la liste non exhaustive des modules CSS3 (tableau 1.1), avec leurs niveaux de validation actuels, en mars 2012 sur lesquels le W3C travaille actuellement.

URL : http://www.w3.org/Style/CSS/current-work

Tab. 1.1 Modules CSS3

Recommendation	
CSS Color level 3	REC
CSS Namespaces	REC
Selectors Level 3	REC
Stable	
Media Queries	CR
CSS Style Attributes	CR
À tester	
CSS Background and Borders Level 3	CR
CSS Marquee	CR
CSS Multi-column Layout	CR
CSS Speech	CR
CSS Mobile Profile 2.0	CR
CSS TV Profile 1.0	CR
Amélioration	
CSS Image Values and Replaced Content Level 3	LC
CSS Transforms	WD
CSS Transitions	WD
CSS Values and Units Level 3	LC
CSS Print Profile	LC

Révision	
CSS Animations	WD
CSS Flexible Box Layout	WD
CSS Fonts Level 3	WD
CSS Paged Media Level 3	LC
CSS Text Level 3	WD
CSS Basic User Interface Level 3	CR
CSS Writing Modes Level 3	WD
CSSOM View	WD
Exploration	
CSS Conditional Rules Level 3	WD
CSS Device Adaptation	WD
CSS Exclusions and Shapes	ED
CSS Generated Content for Paged Media	WD
CSS Grid Layout	WD
CSS Lists Level 3	WD
CSS Positioned Layout Level 3	WD
CSS Regions	WD
CSS Template Layout	WD
Selectors Level 4	WD
CSS Object Model	WD
CSS Fragmentation Level 3	WD

1.4. Les navigateurs

Pour pouvoir utiliser CSS, il nous faut un interpréteur. La plupart du temps, nous l'appelons

navigateur ou encore butineur. Le cœur de ces navigateurs, qu'ils soient *desktop* ou *mobile*, est le fameux moteur de rendu (*Layout Engine*). Mais quel est le rôle de ce moteur ? La réponse est simple : il fait tout ou pratiquement tout (en ce qui concerne les CSS en tout cas). Le navigateur, c'est juste l'intermédiaire entre nous et le moteur. L'interface graphique est d'ailleurs appelée « Chrome » dans certains navigateurs. Ce terme n'a rien à voir avec le butineur du même nom mais désigne le côté esthétique de la matière brillante.

Le moteur est donc responsable de l'affichage de la page et si deux navigateurs implémentent la même version du même moteur, le rendu de la page sera à très peu de chose près identique. C'est pour cette raison que vous verrez dans ce livre aussi bien des références à un navigateur qu'à un moteur (selon la rime !).

En 2012 en France, la navigation internet « classique » se répartit entre cinq navigateurs principaux (figure 1.5) et parmi ces navigateurs (et les autres) seuls quatre moteurs de rendus se détachent et permettent un rendu correct :

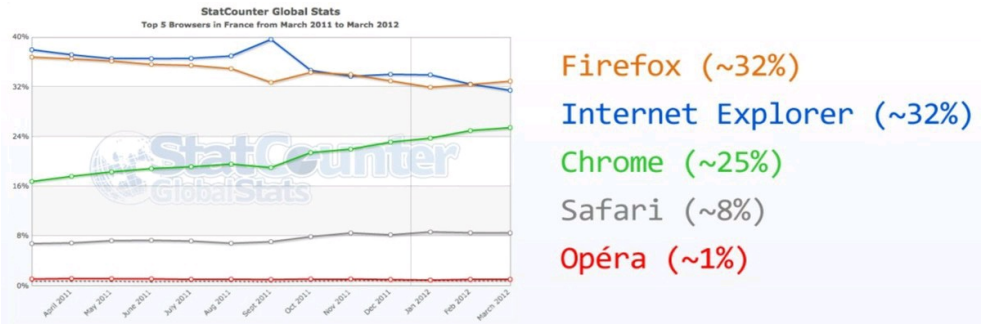


Fig. 1.5 Statistiques de la navigation web en France entre mars 2011 et mars 2012 (source : statcounter.com)

- **Internet Explorer** : c'est le navigateur « emblématique » développé par Microsoft qui a gagné la guerre des navigateurs de la fin des années 1990 face à Netscape Navigator. Ce navigateur est actuellement en version 9 depuis 2011. Cette version a dépassé IE8 en termes d'utilisation en mars 2012 en France. Le moteur de ce navigateur est Trident. Les mises à jour ne sont pas (encore) automatiques.
- **Firefox** : ce navigateur est développé par Mozilla. Depuis Firefox 4 disponible fin mars 2011, la version 2 du moteur est utilisée. Le navigateur propose actuellement des mises à jour automatiques. Le moteur de ce navigateur est Gecko.
- **Chrome** : ce navigateur développé par Google est le plus récent. Les mises à jour sont automatiques. Le moteur de ce navigateur est Webkit qui est issu du moteur libre KHTML.
- **Safari** : ce navigateur est développé par Apple. Les mises à jour sont automatiques. Le moteur de ce navigateur est Webkit, comme pour Chrome.
- **Opéra** : ce navigateur est développé par Opéra. Le moteur de ce navigateur est Presto. Les mises à jour sont automatiques.

Fig. 1.6 Les logos des navigateurs

Concernant la navigation mobile en France, le marché est réparti principalement entre deux

navigateurs (figure 1.7) : Safari Mobile (iPhone, iPad, iPod) et Android Browser (Android). Ces deux navigateurs utilisent le même moteur : Webkit. D'autres navigateurs sont pourtant présents. C'est le cas d'Opéra Mobile et Mini, Firefox Mobile, Chrome Mobile, principalement pour Android, ainsi que des navigateurs dépendant de leurs systèmes d'exploitation, comme IE Mobile ou Blackberry.

Fig. 1.7 Statistiques de la navigation mobile en France entre mars 2011 et mars 2012 (source : statcounter.com)

Tout au long de votre lecture, nous vous proposons des tableaux de compatibilités pour la plupart des fonctionnalités présentées dans ce livre. Ces tableaux indiquent la version des navigateurs à partir de laquelle la fonctionnalité est prise en charge, en précisant si les préfixes sont nécessaires. Le navigateur Opéra mini fonctionnant différemment, nous n'avons pas souhaité le présenter dans ces tableaux. Les déclinaisons mobiles de Chrome et de Firefox ont un support identique à leur version « desktop » équivalente, nous les avons donc omis dans le tableau de compatibilité. La figure 1.8 présente un exemple factice pour la règle inexistante display:holographic.

display: holographic	99 -ms-	99 -moz-	99	99	N.C		99	99 -o-	99

Fig. 1.8 Tableau de compatibilité de display:holographic

1.5. Utiliser CSS3

S'il est un constat quand on regarde un site de promotion des feuilles de styles comme le CSS ZEN GARDEN de Dave Shea, c'est bien celui que le designer web aime ce qui est beau et il faut l'admettre, faire quelque chose de beau avec un border: 4px ridge red; ce n'est pas aisé. La grande majorité des designers a donc recours à des images de substitution pour arriver à reproduire un effet, là où CSS n'est plus capable de leur rendre ce service. C'est sûrement l'un des constats qui a été fait au vu des nombreuses améliorations de CSS3 portant sur la partie purement graphique. Mais CSS3, ce n'est pas que cela. La spécification prévoit de nombreux ajouts, avec notamment des propositions de nouveaux modèles de positionnement ou de gestion du flux, un plus grand contrôle de la gestion de la typographie, l'amélioration de l'expérience utilisateur avec les transitions, les animations et les transformations, la gestion ciblée des médias alternatifs, etc. Ce sont tous ces sujets que nous approfondirons ensemble dans cet ouvrage.

Une question récurrente se pose néanmoins : « *Est-il possible d'utiliser CSS3 aujourd'hui, sur des sites en production et qui s'adressent au plus grand nombre ?* » Notre réponse est oui, même si certaines précautions doivent êtres prises.

Pour le comprendre, commençons par découvrir un concept qui ne provient pas du web, mais qui est

primordial pour pouvoir utiliser CSS3 aujourd'hui : le principe de **dégradation gracieuse** ou d'**amélioration progressive**. Nous verrons ensuite plusieurs méthodologies pour la mise en œuvre de ces principes.

1.5.1. Dégradation gracieuse / amélioration progressive

Sur le web, la dégradation gracieuse ou l'amélioration progressive, *graceful degradation* ou *progressive enhancement* en anglais, sont deux concepts identiques qui reposent sur une idée simple et logique : un site doit pouvoir être visualisé et utilisé quel que soit le support du navigateur. Le site doit continuer à fonctionner même si les techniques employées ne sont pas reconnues dans un navigateur ayant été développé bien avant l'apparition de celles-ci. De la même façon, il n'est pas indispensable que votre site ait rigoureusement le même aspect sur l'ensemble des navigateurs. C'est tout simplement impossible. En revanche, il serait malheureux qu'un contenu important soit rendu inaccessible à cause d'une ombre rendue opaque par un navigateur auquel vous n'auriez pas pensé par exemple.

Là où les deux concepts se divisent, c'est sur la façon de penser. La dégradation gracieuse part du principe qu'un site peut être construit avec toutes les technologies récentes et qu'ensuite une phase de compatibilité doit être entreprise pour un support des anciens navigateurs. L'amélioration progressive, c'est l'inverse. Le principe est de créer un site qui soit compatible avec l'ensemble des navigateurs et la phase d'amélioration qui arrive ensuite apporte une plus-value pour les navigateurs compatibles.

Dans les faits, les deux concepts seront mis en place, selon les fonctionnalités de CSS utilisées (figure 1.9).

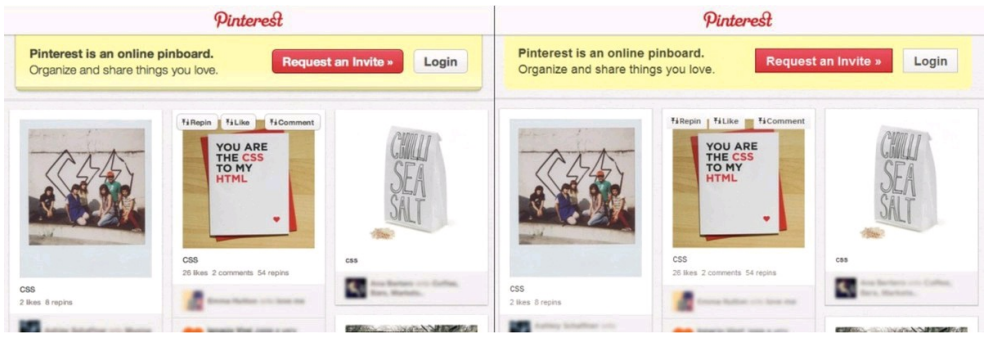


Fig. 1.9 Exemple de différence de rendu entre deux navigateurs sur pinterest.com

Lorsque nous utilisons les propriétés visuelles par exemple, comme pour l'ajout d'ombres ou de coins arrondis, nous préférons parler de dégradation gracieuse. En effet, ces décorations n'étant pas primordiales, leur rendu n'est pas obligatoire tant que le contenu du site reste accessible. De plus, dans le cas où nous souhaitons quand même obtenir un rendu visuel approchant sur des navigateurs non compatibles, nous verrons qu'il est très facile d'appliquer des correctifs ou des solutions de repli. Il est vrai que la dégradation gracieuse fait souvent allusion aux effets graphiques même si ce n'est pas uniquement cela.

D'un autre côté, comme nous allons le voir tout au long de cet ouvrage, CSS3 nous permet d'aller bien plus loin que la simple mise en forme « classique ». Nous pouvons aujourd'hui utiliser des techniques poussées comme les transformations géométriques en 2D et en 3D, les animations, les effets typographiques ou encore les modèles de positionnement avancés. Dans ce cas-là, nous parlons plutôt

d'amélioration progressive. Notre feuille de styles doit alors valoriser notre contenu pour les navigateurs compatibles.

1.5.2. Mise en œuvre

Pour mettre en place ces deux principes, plusieurs actions peuvent être réalisées :

- **Ajout des préfixes vendeurs :** cet aspect est indispensable pour l'utilisation de propriétés CSS3 non encore finalisées. C'est la première étape pour obtenir un rendu similaire sur les navigateurs compatibles
- **Utilisation de styles alternatifs** : dans le cadre de la dégradation gracieuse, il faut alors ajouter des styles alternatifs, pour les navigateurs qui ne comprennent pas certaines fonctionnalités. Nous verrons plusieurs exemples.
- **Détection de fonctionnalités** : pour ce qui est de l'amélioration progressive, le principe est d'appliquer des styles uniquement sur les navigateurs qui les comprennent. Pour cela, nous allons devoir passer par des méthodes de détection de fonctionnalités.

Ne rien faire

Dans le cadre de la mise en place de la dégradation gracieuse ou de l'amélioration progressive, ne rien faire peut être la seule action à entreprendre.

En effet, pour l'ajout de coins arrondis sur une boîte par exemple, il n'est pas nécessaire de prévoir une solution de recours car la vocation n'est que décorative. Cela fait partie du principe de dégradation gracieuse.

Dans le même temps, lors d'ajout de fonctionnalités nouvelles de CSS (comme les transitions, les animations) celles-ci ne poseront aucun problème de compatibilité par les navigateurs trop anciens puisque, n'étant pas reconnues, elles ne seront pas interprétées. Il faut bien entendu prendre soin à ce que tout fonctionne normalement avant l'ajout de ces fonctionnalités: c'est donc le principe d'amélioration progressive.

Bien que cela puisse suffire dans certains cas, d'autres actions peuvent/doivent êtres entreprises. C'est notamment le cas pour l'utilisation des préfixes.

Utilisation des préfixes

Nous l'avons vu précédemment, les propriétés préfixées sont recommandées tant que le brouillon de spécification n'a pas atteint le stade de recommandation. Chaque moteur de navigateur utilise alors son propre préfixe, afin de limiter les interférences entre eux. Voici les différents préfixes connus :

- -ms- pour Microsoft (Internet Explorer, IE Mobile) ;
- -moz- pour les produits Mozilla (Firefox, Firefox Mobile) ;
- -webkit- pour Chrome, Safari, leurs versions mobiles et en règle générale tous les navigateurs basés sur le moteur Webkit ;
- -o- pour les navigateurs Opéra (Opéra, Opéra Mobile, Opéra Mini) ;
- quelques autres anecdotiques.

L'une des étapes pour profiter des fabuleuses propriétés CSS3 sur un site web est d'utiliser les différentes versions préfixées de ces propriétés, car la majorité des modules n'ont pas encore atteint le statut de recommandation. Cela nous permet d'offrir un support au sein de tous les navigateurs qui implémentent ces propriétés « brouillons ».

Pour ce faire, nous nous devons d'écrire la même propriété CSS3 plusieurs fois. Par exemple, dans le cas d'une transformation d'un élément dans l'espace, la propriété transform doit être écrite cinq fois, comme ceci :

Utilisation des préfixes propriétaires

```
div{
    -webkit-transform: rotateX(45deg);
       -moz-transform: rotateX(45deg);
        -ms-transform: rotateX(45deg);
         -o-transform: rotateX(45deg);
            transform: rotateX(45deg);
}
```

Dans cette déclaration, les navigateurs basés sur Webkit comprennent la première écriture, ceux basés sur Gecko comprennent la deuxième, Internet Explorer comprend la troisième et les navigateurs Opéra comprennent la quatrième. Nous terminons par l'écriture « normée », alors même qu'aucun navigateur n'implémente actuellement cette règle. D'ailleurs, il en est de même pour l'ajout de la propriété préfixée d'Opéra qui ne supporte pas encore les transformations 3D. Cependant, l'ajout de toutes les déclarations préfixées, en terminant par la version standard, est indispensable. De cette façon, notre code devient évolutif. Ainsi, si la spécification passe en recommandation et que les navigateurs commencent à supporter la version non préfixée, où si, dans notre cas, Opéra commence à supporter les transformations 3D, notre code CSS sera déjà prêt. Nous n'aurons pas besoin de parcourir à nouveau toutes nos feuilles de styles pour ajouter ces propriétés.

Cela peut éventuellement poser d'autres problèmes, notamment dans le cas où la spécification évoluerait avec un changement de syntaxe. Nous ne sommes pas à l'abri d'avoir un mauvais rendu si notre CSS utilise une ancienne syntaxe, soit disant normée. Mais c'est le jeu ma pauv'Lucette ! Nous vous conseillons néanmoins cette écriture, surtout si la propriété s'approche de la recommandation.

Continuons avec une bonne nouvelle pour nous. Dans certains cas, les propriétés n'ont pas besoin d'être écrites autant de fois que de navigateur. C'est le cas notamment de box-shadow, qui permet d'ajouter des ombres sur une boîte.

```
div{
    -webkit-box-shadow: 0 0 5px black;
       -moz-box-shadow: 0 0 5px black;
            box-shadow: 0 0 5px black;
}
```

Nous remarquons que dans ce cas précis, l'ajout des préfixes -ms- et -o- a été volontairement omis. Tout simplement car ces deux navigateurs n'ont jamais supporté box-shadow en version préfixée. Cette propriété a eu un support immédiat dans Opéra 10.5 et Internet Explorer 9, les éditeurs estimant que cette propriété ne serait plus modifiée. Depuis, Firefox, Chrome et Safari ont suivi le pas.

Cependant, comme expliqué juste avant, l'ajout de tous les préfixes connus ainsi que l'ordre d'écriture est très important. En effet, il est conseillé de toujours terminer sa déclaration par la version non préfixée d'une propriété. Ainsi, les navigateurs compatibles avec deux écritures utiliseront la dernière de la liste, en suivant le principe de la cascade CSS. Cette dernière étant censée être normalisée.

Terminons notre tour d'horizon de l'utilisation des préfixes par une encore meilleure nouvelle.

Quelques propriétés CSS3 n'ont besoin d'aucuns préfixes. C'est le cas par exemple de text-shadow, d'opacity ou encore de word-wrap. La raison est toute simple, ces propriétés étaient présentes dans la spécification CSS2 qui n'a jamais vu le jour ou ont été créées par les navigateurs eux-mêmes il y a quelques années et proposées au W3C ensuite pour être ajoutées à la norme. Certains navigateurs avaient donc déjà commencé à les supporter.

Pour plus de clarté, tous les exemples de code présents dans le livre sont **dépourvus de préfixes**. Nous ne le précisons pas à chaque fois. En revanche, un tableau de compatibilité pour chaque propriété, avec l'ajout ou non des préfixes, vous informe de la syntaxe à utiliser. De plus, l'écriture des préfixes peut être automatisée à l'aide d'outils présentés dans le chapitre dédié.

Styles alternatifs

Pour les navigateurs qui ne supportent pas les fonctionnalités définies dans les spécifications CSS3, il nous faut quelques fois trouver des astuces pour maintenir une dégradation gracieuse. Il existe pour cela plusieurs méthodes spécifiques à chaque fonctionnalité que nous souhaitons reproduire. Cela passe principalement par des règles CSS différentes ou conditionnelles.

Prenons l'exemple des fonctions de couleurs avec transparence de CSS3, comme rgba(). Cette fonction n'est malheureusement pas reconnue dans les navigateurs trop anciens. Lors de l'utilisation, nous devons alors prévoir une deuxième déclaration identique avec une valeur de couleur reconnue par l'ensemble des navigateurs. Cette déclaration doit être ajoutée avant la notation CSS3, comme ceci :

```
div{
	color: red;
	color: rgba( 255, 0, 0, 0.5);
}
```

Dans ce cas, la deuxième valeur de couleur est utilisée par les navigateurs compatibles, en suivant le principe de la cascade CSS. Pour les autres, les déclarations non comprises sont tout simplement ignorées. Cela signifie que c'est la première déclaration color: red; qui est utilisée. Sans cette déclaration, les navigateurs non compatibles avec rgba() n'appliquent aucune couleur.

Cette astuce est très utilisée pour de nombreuses propriétés CSS3, comme par exemple pour les dégradés, où nous devons prévoir une couleur unie ou une image comme solution de repli pour la dégradation gracieuse. Ces techniques sont détaillées dans les chapitres dédiés.

Concernant les anciennes versions d'Internet Explorer qui posent problème en CSS, la méthodologie peut quelques fois être différente. En effet, certains styles doivent être appliqués exclusivement à certaines versions du navigateur. Nous utilisons alors les feuilles de styles ou les classes conditionnelles, *via* les commentaires conditionnels. Ce mécanisme, uniquement disponible sur le navigateur Internet Explorer jusqu'en version 9, permet de limiter l'interprétation d'une partie du code HTML à ces navigateurs. La syntaxe repose sur les commentaires classiques HTML, lesquels sont enrichis de critères propres au navigateur, sous la forme d'un test. La syntaxe est la suivante pour un commentaire conditionnel ciblant toutes les versions :

```
<!–[if IE]>
<p>Code HTML exclusif à IE</p>
<![endif]–>
```

Afin d'être encore plus précis dans le ciblage, les mots-clés suivants sont reconnus: lt (*less than* :

moins que), gt (*greater than* : plus que), lte (*less than equal* : moins ou égal à), gte (*greater than equal* : plus ou égal à) ainsi que les caractères & (et logique), | (ou logique) et ! (négation logique). Voici des exemples d'écriture de ces commentaires :

```
<!–[if IE 8]>     ...  Code HTML exclusif à IE 8 <![endif]–>
<!–[if lt IE 8]>  ...  Code HTML exclusif à IE 7 et inférieurs <![endif]–>
<!–[if lte IE 8]> ...  Code HTML exclusif à IE 8 et inférieurs <![endif]–>
<!–[if gte IE 8]> ...  Code HTML exclusif à IE 8 et supérieur <![endif]–>
<!–[if !(IE 8)]>  ...  Code HTML exclusif à IE sauf IE 8 <![endif]–>
```

Grâce à ces commentaires conditionnels, nous allons pouvoir charger une feuille de style conditionnelle, uniquement pour les navigateurs qui demandent des correctifs, comme ceci :

```
<link rel="stylesheet" type="text/css" href="style.css" media="screen" />
<!–[if lte IE 7]>
   <link rel="stylesheet" type="text/css" href="styleIE.css" media="screen" />
<![endif]–>
```

Dans ce cas, la feuille de styles est chargée pour les versions d'IE égale ou inférieure à 7. Cette feuille de styles est généralement appelée après la feuille de styles standard, afin de pouvoir surclasser les règles.

Cette technique, si efficace soit-elle, pose plusieurs soucis :

- Les navigateurs doivent charger d'autres fichiers, ce qui induit plusieurs requêtes HTTP. De plus, ces versions ciblées étant déjà anciennes, le temps de chargement d'une page est alors très conséquent.
- C'est souvent plusieurs fichiers CSS qui devront êtres ajoutés, en fonction des bugs de versions. Nous nous retrouvons alors très facilement avec un fichier pour IE8, un autre pour IE8 et 7, et un autre pour IE7. Le maintient de ces fichiers est alors plus difficile. Chaque modification de styles entraîne des vérifications supplémentaires.

C'est pourquoi une alternative à ces feuilles de styles conditionnelles existe. Cette nouvelle solution a été initialement proposée par Paul Irish sur son blog et fait depuis l'unanimité au sein de la communauté web. C'est par exemple la solution retenue pour HTML5 BoilerPlate.

Sa solution repose également sur les commentaires conditionnels, mais ceux-ci sont utilisés pour l'ajout de classes conditionnelles sur une balise parent du document, comme html. Cela nous donne le code suivant :

```
<!DOCTYPE html>
<!–[if lt IE 7]> <html class="lt-ie9 lt-ie8 lt-ie7"> <![endif]–>
<!–[if IE 7]>    <html class="lt-ie9 lt-ie8"> <![endif]–>
<!–[if IE 8]>    <html class="lt-ie9"> <![endif]–>
<!–[if gt IE 8]><!–> <html class=""> <!–<![endif]–>
```

Commentaires conditionnels

Cette syntaxe est celle préconisée en date du 17 janvier 2012, après trois ans d'itération successive. Retrouvez l'historique (passé et futur) sur le blog de Paul Irish.

URL : http://paulirish.com/2008/conditional-stylesheets-vs-css-hacks-answer-neither/

Grâce à cette écriture conditionnelle, il devient facile d'appliquer des styles alternatifs pour des

versions d'IE précises, *via* le sélecteur de classe et le sélecteur d'enfant :

```
#elem{
	box-shadow: 0 0 5px black;
}
.lt-ie9 #elem{
	border: 1px solid black;
}
```

Dans ce cas, la propriété box-shadow, n'étant pas reconnue avant IE9, est « remplacée » par une bordure dans les navigateurs IE8 et inférieurs. Si cette bordure avait été ajoutée dans le même bloc de déclaration que l'ombre portée, tous les navigateurs l'auraient ajoutée en plus de l'ombre (figure 1.10).

Fig. 1.10 Dégradation gracieuse conditionnelle

Détection de fonctionnalités

Dans le cadre de la mise en place de l'amélioration progressive, il est envisageable d'ajouter certains styles CSS pour les navigateurs compatibles uniquement. Pour ce faire, deux techniques existent :

- **La détection de navigateurs** (*browser sniffing*) : cette technique très prisée il y a quelques années permet de récupérer le nom et la version d'un navigateur. Elle est totalement à proscrire aujourd'hui pour plusieurs raisons. La première raison évidente est que le maintient d'une liste de fonctionnalités supportées pour chaque version de navigateur est difficile, surtout avec l'évolution de ceux-ci qui est de plus en plus fréquente. Une autre raison est sans nul doute liée aux problèmes de la détection elle-même. En effet, celle-ci repose sur le mécanisme des expressions régulières, qui ne sont pas simples à mettre en place et très facilement soumises aux bugs dans le temps. Pour faire simple, cette technique est obsolète.
- **La détection de fonctionnalités** (*features detection*) : c'est la solution retenue actuellement par la communauté web. Le principe est de demander au navigateur si une fonctionnalité est supportée ou non. Bien que CSS3 définisse la règle @supports pour cela, la technique aujourd'hui passe principalement par un langage de script côté client, tel JavaScript, cette @-règle n'étant implémentée nulle part. Nous pourrons alors utiliser des librairies comme Modernizr pour nous aider dans notre quête de l'amélioration progressive.

Prenons l'exemple des transformations CSS. Celles-ci ne sont supportées que par un nombre limité de navigateurs. Le principe de l'amélioration progressive s'applique donc ici parfaitement. Avec l'aide de Modernizr, dont le principe est d'ajouter des classes conditionnelles sur la balise html en fonction de ce que supporte le navigateur, nous allons facilement appliquer nos transformations sans risques :

```
html.csstransform3d div{
	transform: rotateY(20deg);
}
```

Dans ce cas, l'élément div dont le parent html à la classe .csstransform3d sera transformé. Les navigateurs non compatibles n'auront pas la classe .csstransform3d et donc ce style ne sera pas appliqué. L'utilisation détaillée de Modernizr est expliquée dans le chapitre consacré aux outils. Retenez quand même déjà que cette solution requiert JavaScript.

Ce qu'il faut retenir...

Ce tour d'horizon de la norme CSS, avec son principe modulaire, vous permet de répondre à l'une des questions très fréquemment posée: oui, CSS3 est utilisable dès aujourd'hui sur la plupart des sites web, même si ceux-ci s'adressent à un public très large. C'est d'ailleurs ce que font la majorité des sites web de grande envergure, comme Google avec la refonte visuelle complète de tous ses services (moteur de recherche, Google Plus, Youtube, etc.), Twitter dont la nouvelle interface date de début 2012, Facebook avec sa nouvelle timeline ou encore Microsoft et le nouveau Windows 8 Metro où CSS3 s'impose comme le langage de présentation des données applicatives.

En revanche, il faut toujours avoir en tête que CSS3 doit être vu comme une gigantesque ville en construction où les modules peuvent êtres assimilés à des quartiers. Certains quartiers sont terminés et habitables dès aujourd'hui, d'autres le seront bientôt, et d'autres encore n'ont toujours pas vu le jour.

Les éléments du langage

Objectifs

CSS3 apporte son lot de nouveautés tant sur le point graphique que technologique. L'officialisation d'une nouvelle fonctionnalité ou d'une nouvelle propriété par le W3C ne se fait pas à la légère. Elle peut être initiée par les éditeurs de navigateurs, qui en implémentant une nouveauté proposent son intégration dans la norme si celle-ci est bien accueillie des développeurs.

Du sélecteur au curseur, nous allons maintenant faire le point sur les éléments constituant ce langage. Nous en profiterons pour voir ou revoir certaines fonctionnalités CSS2.1 peu utilisées jusqu'à présent et pourtant bien supportées par les navigateurs.

2.1. Principe de CSS

Avant de rentrer dans le vif du sujet et de plonger au cœur des nouveautés de CSS3 il nous semblait important de rappeler les bases du fonctionnement des feuilles de styles en cascade.

Les feuilles de styles CSS sont composées de blocs définissant un ensemble de déclarations à appliquer à une ou plusieurs balises HTML dans le but de définir la présentation de la page web. Ces balises sont désignées par le sélecteur CSS précédant le bloc.

Au chargement de la page HTML, le navigateur construit un modèle arborescent correspondant à l'ensemble des balises de la page web appelé le DOM (*Document Object Model*). Il charge ensuite la (ou les) feuille(s) de styles puis applique les différentes règles aux objets du DOM au fur et à mesure de la lecture des styles.

Les styles peuvent êtres écrits dans l'attribut style d'une balise quelconque, dans une balise style en en-tête de la page HTML ou bien dans un fichier séparé qui sera lié à la page de différentes manières, avec une balise link par exemple.

Intégration dans un attribut de balise HTML

```
<p style="text-decoration: underline;">BugDroid est mon ami</p>
```

Cette méthode d'inclusion des styles permet de modifier un seul élément de la page. Cette une méthode non optimisée et non préconisée. Cependant, dans certains cas particuliers (code généré depuis un CMS, support de client email, API JavaScript, styles dynamiques), cette notation peut se révéler utile. Son utilisation est cependant dangereuse car les styles ainsi définis sont prioritaires sur les styles définis avec les autres méthodes que voici.

Intégration dans la page HTML

```
<style type="text/css">
	body{
		color: navy;
```

```
    }
</style>
```

Cette syntaxe a l'avantage de limiter le nombre de requêtes HTTP. En revanche elle est moins intéressante dès lors que notre site contient plusieurs pages, la partie style étant chargée de nouveau à chaque requête. C'est donc une méthode qui peut se révéler intéressante dans des cas spécifiques, comme un site à page unique ou dans le cas d'optimisations de requêtes lors de la création de pages dynamiques, depuis un langage coté serveur.

Appel d'une feuille de style située dans le dossier chemin depuis la balise link

```
<link rel="stylesheet" type="text/css" href="chemin/style.css" />
```

Appel d'une feuille de style avec la règle @import depuis la page HTML

```
<style type="text/css">
    @import url(chemin/style.css);
</style>
```

Ces deux dernières méthodes sont sensiblement identiques. Elles permettent de gérer une feuille de styles externe qui est liée à plusieurs pages HTML, ce qui permet de limiter le téléchargement des styles par chacune des pages de notre site. De plus, la définition du média (écran, impression...) pour lequel la feuille de styles s'applique peut être ajoutée.

Voici un exemple simple d'utilisation d'une feuille de styles CSS dans une page HTML 5 (tableau 2.1 et figure 2.1).

Tab. 2.1 Utilisation d'une feuille de styles CSS dans une page HTML 5

HTML (index.html)	CSS (chemin/styles.css)

<!DOCTYPE html>
<html>
<head>
<title>CSS3</title>
<meta charset="utf-8" />
<link rel="stylesheet" type="text/css" media="screen" href="chemin/style.css" />
</head>
<body>
 CSS addict!
</body>
</html>
	h1{
	background-color: navy;
	color: aquamarine;
	text-align: center;
	}

Fig. 2.1 Rendu visuel

Pourquoi « *Cascading Style Sheet* » ? Que viennent faire les cascades dans tout ceci ? Le concept est tout simple. La majorité des styles que l'on applique à une balise HTML va être propagée à ses enfants (les balises incluses), ainsi qu'à tous les descendants, ce qui forme une cascade. Donc, si nous définissons la couleur rouge pour le texte du body, les balises h1, p et autres représenterons leur texte en rouge jusqu'à ce que nous n'indiquions le contraire.

Certains styles sont définis dans le navigateur lui-même (l'aspect des liens par exemple) et ces styles sont lus avant les nôtres, ce sont les styles par défaut.

Le CSS n'est donc pas vraiment un langage informatique mais plutôt une syntaxe régie par des règles de grammaires informatiques assez strictes, ceci dans le but de limiter le travail du navigateur pour parser (traduire) votre code CSS.

2.2. Les sélecteurs

Les sélecteurs sont la base de la recherche d'éléments HTML dans le DOM. Leur utilisation combinée est la clé d'une feuille de styles optimisée, il est donc important de bien les connaître. La présentation de cette liste de sélecteur se veut la plus exhaustive possible, la spécification *Selectors CSS3* du W3C étant une recommandation.

Comme un petit rappel ne fait jamais de mal, voici la syntaxe d'un sélecteur CSS nommé sélecteurCSS permettant de définir la valeur valeurCSS pour la propriété propriétéCSS des éléments HTML ciblés par ce sélecteur :

```
sélecteurCSS {
	propriétéCSS: valeurCSS;
}
```

Nous vous présentons ici tous les sélecteurs CSS classés par ancienneté avec les cas d'utilisations des nouveaux sélecteurs, ainsi que des exemples montrant les particularités de certains.

Chaque sélecteur est suivi de sa description qu'il faudra lire ainsi :
ce sélecteur *désigne* un élément quelconque.

2.2.1. Les sélecteurs CSS1

Voici la liste de ces sélecteurs pour lesquels nous ne rentrerons pas dans le détail :

- E un élément de type E ;
- E F un élément F descendant d'un élément E ;
- #foo un élément ayant pour attribut id="foo" ;
- .bar un élément ayant pour attribut class="bar" ;
- E,F les éléments E et F.

2.2.2. Les sélecteurs CSS2.1

Dont certains sont trop souvent oubliés :

- * n'importe quel élément ;
- E>F un élément F, descendant direct d'un élément E.
  Ce sélecteur permet par exemple de cibler un élément de premier niveau du body de la page

HTML (tableau 2.2 et figure 2.2).

Tab. 2.2

HTML	CSS

<body>
<header>CSS</header>
<section>
<header>est</header>
 fabuleux
</section>
</body>
	body > header{
	font-size: 2em;
	color: aquamarine;
	}

CSS
est
fabuleux

Fig. 2.2 Rendu visuel

- E+F un élément F successeur direct d'un élément E (tableau 2.3 et figure 2.3)

Tab. 2.3

HTML	CSS

 CSS
 est
 fabuleux
 	
	h2 + p{
	color: aquamarine;
	}

CSS
est
fabuleux

Fig. 2.3 Rendu visuel

- E[foo] un élément avec un attribut foo ;
- E[foo="bar"] un élément dont l'attribut foo a exactement la valeur bar ;
- E[foo~="bar"] un élément dont l'attribut foo a au moins la valeur bar.
  Ce sélecteur est surtout utile pour cibler des attributs prenant plusieurs valeurs comme l'attribut class. Les sélecteurs suivants sont donc rigoureusement identiques (tableau 2.4).

Tab. 2.4

HTML	CSS

Bugdroid aime les carottes
 	
	div[class~="droid"]{
	color: orangered;
	}
	div.droid{
	color: orangered;
	}

- E[foo|="bar"] un élément dont la valeur de l'attribut foo est égale à une valeur commençant par bar-.
  Ce sélecteur est principalement utilisé avec l'attribut lang pour tester les déclinaisons de celui-ci. Ex. : *[lang|=en] cible pour l'attribut lang les valeurs en-US, en-GB, etc.

2.2.3. Les sélecteurs CSS3

- E~F un élément F qui est précédé d'un frère de type E (pas forcément juste avant, voir tableau 2.5 et figure 2.4).

Tab. 2.5

HTML	CSS

 CSS
 est
 totalement
 fabuleux
 	
	h2 ~ p{
	color: blueviolet;
	}

Fig. 2.4 Rendu visuel

- E[foo□="bar"] un élément E dont la valeur de l'attribut foo commence par bar.
  Ce sélecteur permet par exemple la sélection d'un groupe de balises ayant un identifiant de racine commune, sans devoir ajouter d'attribut class.

Tab. 2.6

HTML	CSS

CSS
 est
 Fabuleux
	div[id□="partie"]{
	text-decoration: underline;
	}

- E[foo$="bar"] un élément E dont la valeur de l'attribut foo se termine par bar.
- E[foo*="bar"] un élément E dont la valeur de l'attribut foo contient bar.

Tab. 2.7

HTML	CSS

Mi-Humain
Mi-Android
	span[class*="oid"]{
	color: yellowgreen;
	}

Sélecteurs CSS3	8.0	3.5	4.0	3.2	9.5	3.2	10.0	2.1

Fig. 2.5 Tableau de compatibilité des sélecteurs CSS3

2.2.4. Les pseudo-classes CSS1 et CSS2.1

Un peu particuliers, les sélecteurs de pseudo-classes permettent de caractériser le sélecteur principal. Ils se combinent à un sélecteur simple ou composé. Ils peuvent s'appliquer à l'élément ciblé ou à un des parents dans le sélecteur hiérarchique. L'écriture suivante est autorisée : body p:hover a{ } et cible l'ensemble des liens d'un paragraphe au survol de ce dernier.

- E:first-child un élément E qui est le premier enfant de son parent.

Parmi les sélecteurs de pseudo-classes on peut distinguer les sélecteurs suivants qui permettent d'interagir avec l'utilisateur, soit par le pointeur de la souris, soit par le focus du clavier :

- E:active un élément E activé par le pointeur (durant le clic) ;
- E:hover un élément E survolé par le pointeur ;
- E:focus un élément E ayant le focus (clavier ou souris).

S'il s'agit d'un lien, la validation par la touche Entrée activera le lien, s'il s'agit d'une case à cocher, la touche Espace la fera basculer.

- E:link un élément E dont le lien n'a pas encore été visité ;
- E:visited un élément E dont le lien a déjà été visité ;

- E:lang(fr) un élément E en langue française. (si l'attribut lang="fr" est défini dans la balise html).

2.2.5. Les pseudo-classes CSS3

- E:root un élément E qui est la racine du document. En HTML il s'agit toujours de html.
- E:nth-child(n) un élément E qui est le $n^{ième}$ enfant de son parent, quel que soit le type de cet enfant. La valeur de n peut être un entier ou bien l'expression $an+b$ avec a et b entiers relatifs pour toute valeur de $n >= 0$ (tableau 2.8 et figure 2.8). Pour effectuer la sélection, on divise l'ensemble des balises E en groupes de a éléments et on retient le $b^{ième}$ de chaque groupe.
  Ce sélecteur et ses dérivés sont très puissants, il est en effet possible d'effectuer des cycles particulièrement intéressants pour la mise en forme des menus. Il existe deux mots clés even et odd et qui permettent directement la sélection des éléments pairs ou impairs.
  :nth-child(10n+5) : les 5ème, 15ème, 25ème, ... éléments.
  :nth-child(2n+0), :nth-child(2n), :nth-child(even) : les éléments pairs.
  :nth-child(2n+1), :nth-child(odd) : les éléments impairs.
  :nth-child(2n+10) : les éléments pairs à partir de 10.
  Si la valeur a est égale à 0 alors l'écriture nth-child(b) est retenue et le sélecteur cible exclusivement le $n^{ième}$ élément.
  :nth-child(0n+5), :nth-child(5) : le 5ème élément.
  Le sélecteur de pseudo-classe :nth-child ne considère que les valeurs positives (ou égales à zéro) pour la valeur de son paramètre. Cependant, les valeurs a et b peuvent être négatives et seules les valeurs positives du résultat seront prises en compte.

Le mécanisme de :nth-child

:nth-child(-3n+10);

Pour trouver les éléments ciblés, essayons toutes les valeurs de n pour $n >= 0$:
$n = 0 : -3n + 10 = 10$
$n = 1 : -3n + 10 = 7$
$n = 2 : -3n + 10 = 4$
$n = 3 : -3n + 10 = 1$
$n = 4 : -3n + 10 = -2$! cet élément n'est pas pris compte car il n'existe pas.
Ce sélecteur cible donc les 10ème, 7ème, 4ème et 1er éléments.
Voici d'autres exemples typiques :
:nth-child(10n-1) : les 9ème, 19ème, 29ème, … éléments.
:nth-child(-n+6) : les six premiers éléments.
:nth-child(-2n+30) : les éléments pairs jusqu'au trentième.

Tab. 2.8

HTML	CSS

 - Droid
 - Robots
 - Humanoid
 - Cyborg
 - Replicant
 	
	ul li:nth-child(n+3){
	color: coral;
	}

- Droid
- Robots
- Humanoid
- Cyborg
- Replicant

Fig. 2.6 Rendu visuel

Attention : le sélecteur ne fait pas la corrélation entre l'index de l'enfant et le nombre d'éléments E (tableau 2.9 et figure 2.7). Si le parent contient plusieurs types d'éléments, il conviendra d'utiliser le sélecteur nth-of-type décrit plus bas. Dans l'exemple qui suit les deux premières règles sont bien appliquées, en revanche la troisième n'a pas de correspondance dans le DOM, le deuxième élément n'étant pas une balise h2.

Tab. 2.9

HTML	CSS

 CSS
 est
 totalement
 fabuleux
 	
	div p:nth-child(3){
	color: coral;
	}
	div *:nth-child(2){
	font-style: italic;
	}
	div h2:nth-child(2){
	text-decoration: underline;
	}

CSS
est
totalement
fabuleux

Fig. 2.7 Rendu visuel

Ce sélecteur est utilisé dans sa version la plus simple sur le site de démonstration au niveau du diagramme démographique.

- E:nth-last-child(n) un élément *E* qui est le $n^{ème}$ enfant de son parent en partant de la fin. Les règles sont les mêmes que pour :nth-child(n).
- E:nth-of-type(n) un élément E qui est le $n^{ème}$ enfant de type E de son parent. Les règles sont les mêmes que pour :nth-child(n). Ce sélecteur est particulièrement utile pour cibler les *n* premiers éléments d'un type parmi d'autres (tableau 2.10 et figure 2.8).

Tab. 2.10

HTML	CSS

Origine
 Bug
 Aspect
 #99CC33
 Alimentation
 Carottes
	/*Les deux premiers paragraphes */
	p:nth-of-type(-n+2){
	color: coral;
	}

Origine
Bug
Aspect
#99CC33
Alimentation
Carottes

Fig. 2.8 Rendu visuel

- E:nth-last-of-type(n) un élément *E* qui est le $n^{ième}$ enfant de type *E* de son parent en partant de la fin. Les règles sont les mêmes que pour nth-child(n).
- E:last-child un élément *E* qui est le dernier enfant de son parent.
- E:first-of-type un élément *E* qui est le premier enfant de type *E* de son parent (le premier enfant du parent pouvant donc être un autre élément).
- E:last-of-type un élément *E* qui est le dernier enfant de type *E* de son parent.
- E:only-child un élément *E* qui est enfant unique de son parent.

- E:only-of-type un élément *E* qui est le seul enfant de type *E* de son parent. Ce sélecteur permet de traiter différemment les cas où un élément contient un seul ou plusieurs enfants d'un type donné.
- E:empty un élément *E* qui n'a aucun enfant (y compris du texte). Ce sélecteur permet de cibler les balises vides.
- E:target un élément *E* qui est la cible activée d'un hyperlien au sein d'un document (ancre). Ce sélecteur apporte une grande nouveauté dans CSS, il permet de modifier l'aspect d'un élément en fonction d'un évènement d'hyperlien et ceci au sein du document éventuellement sans rechargement de la page (tableau 2.11 et figure 2.9). Son emploi principal est de modifier l'aspect d'une partie de la page lorsque celle-ci est la destination d'un lien. Il est notamment utilisé dans le site de démo pour afficher les petites puces vertes lorsque l'on utilise le menu pour se déplacer dans la page.

Tab. 2.11

HTML	CSS

<nav>
Origines
Contacts
</nav>
<section id="origine">\
 Origines de Bugdroid
</section>
<section id="contact">\
 Contacts sur Bug
</section>
	section:target h2{
	color: coral;
	text-decoration: underline;
	}

Origines Contacts

Origines de Bugdroid

Contacts sur Bug

Fig. 2.9 Rendu visuel

- E:enabled un élément de formulaire E qui est actif.
- E:disabled un élément de formulaire E qui est inactif.
- E:checked un élément de formulaire E qui est coché ou choisi.
- E:indeterminate un élément de formulaire E qui n'a pas de valeur (ex. : groupe de bouton radio dont aucun n'est coché...).

Note : les sélecteurs opérant sur les formulaires ne laissent pas beaucoup de marge de manœuvre au niveau de la mise en forme CSS. Nous abordons légèrement ce point en fin de chapitre.

- E:not(s) un élément E qui ne correspond pas au sélecteur *s*. *s* peut être n'importe quel sélecteur simple excepté :not() (tableau 2.12 et figure 2.10).

Tab. 2.12

HTML	CSS

 CSS
 est
 totalement
 fabuleux
 	
	div *:not(p){
	color: coral;
	}
	div p:not(:first-of-type){
	color: aquamarine;
	}

Fig. 2.10 Rendu visuel

La plupart des cas d'utilisation de cette pseudo-classe sont réalisables avec les sélecteurs « classiques », mais cette syntaxe est souvent plus concise.

Pseudo-classes CSS3	9.0	3.5	4.0	3.2	9.5	3.2	10.0	2.1

Fig. 2.11 Tableau de compatibilité des pseudo-classes

2.2.6. Les pseudo-éléments CSS1 et CSS3

Les sélecteurs de pseudo-éléments ont un mode de fonctionnement différent, ils permettent de sélectionner une partie d'un élément afin de le mettre en forme. Cette sélection n'est pas possible par l'utilisation simple du DOM car rien n'identifie ces éléments dans ce dernier. Historiquement (CSS1), la notation des pseudo-éléments était la même que pour les pseudo-classes, avec le caractère deux-points (:). Afin de différencier les pseudo-éléments des pseudo-classes, la notation double deux-points (::) est introduite. Nous utiliserons donc ici cette dernière notation pour tous les pseudo-élements.

- E::first-line la première ligne formatée d'un élément E.
  Cette ligne n'étant pas délimitée dans l'arbre DOM, c'est le navigateur qui va la délimiter dans l'arbre de rendu (*render-tree*). De l'implémentation de cette fonctionnalité dépendra l'effet lors

de la modification de l'emprise du texte de l'élément E (redimensionnement, augmentation de la police...). Les propriétés pouvant être appliquées à un pseudo-élément ::first-line sont celles concernant directement le texte tel : text-transform, text-decoration, letter-spacing, word-spacing, vertical-align, line-height, ainsi que les propriétés de polices et de couleurs.

- E::first-letter la première lettre d'un élément (tableau 2.13 et figure 2.12).
  Cette lettre peut être accompagnée d'un élément de ponctuation selon les règles d'orthographe de la langue utilisée (l'apostrophe d'élision par exemple en français). La lettre ainsi formatée doit apparaître dans la première ligne de l'élément E. Il peut sembler intéressant d'utiliser la règle float:left; sur le pseudo-élément ::first-letter afin de donner un effet de lettrine. Le pseudo-élément peut aussi désigner la première lettre d'un contenu ajouté *via* content: "Text"; dans un pseudo-élément ::before.

Tab. 2.13

HTML	CSS

CSS est totalement fabuleux ...	p::first-letter{
	float: left;
	color: crimson;
	font-size: 3em;
	line-height: 0.5em;
	}

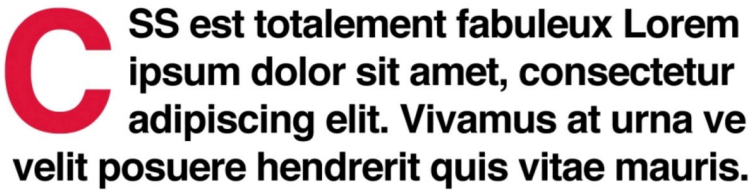


Fig. 2.12 Rendu visuel

- E::selection déprécié. Le texte sélectionné par l'utilisateur.
  Ce sélecteur a été enlevé de la spécification CSS3suite à des problèmes d'interopérabilité. Il permet de styliser la sélection du texte par l'utilisateur. Il est toujours actif en version préfixée ou non dans certains navigateurs, mais à utiliser avec précaution. ::-moz-selection dans Firefox et ::selection dans les navigateurs webkit.
- E::before le contenu éventuellement généré avant l'élément E.
- E::after le contenu éventuellement généré après l'élément E.
  Ces sélecteurs s'utilisent conjointement à la règle content: "Contenu généré" ;. Ils sont utilisés pour générer du contenu supplémentaire dans le rendu de la page HTML à partir des règles CSS. Le contenu généré, inséré avant ou après l'élément, est inclus dans l'espace de boîte de l'élément E. Il permet également d'afficher dans la page le contenu de certains attributs (tableau 2.14 et figure 2.13). Ce pseudo-élément peut aussi être utilisé en remplacement du pseudo-élément ::marker qui n'est pas opérationnel pour le moment.

Tab. 2.14

HTML	CSS

CSS
 Bugdroid
	p{
	text-align:center;
	}
	p::after{
	content: " est " attr(class);
	background: #ddd;
	color: crimson;
	}

CSS est fabuleux

Bugdroid est sympa

Fig. 2.13 Rendu visuel

Ce sélecteur est utilisé sur le site de démo pour générer les infobulles indiquant la démographie sur le graphique.

- E::marker le contenu éventuel de l'élément de liste *E*. Ce pseudo-élément est décrit dans le chapitre sur les listes. (Aucune implémentation à ce jour).

Pseudo-éléments CSS3	9.0	3.5	4.0	3.2	9.5	3.2	10.0	2.1

Fig. 2.14 Tableau de compatibilité des pseudo-éléments
Ce tableau montre le support moyen des pseudo-éléments (avec ::).
IE8 reconnait certains pseudo-éléments avec l'ancienne notation (:).

2.2.7. Le poids des sélecteurs

La spécification précise un poids (traduction de l'auteur de *specificity*) pour chaque sélecteur ainsi qu'une méthode permettant de calculer le poids d'une combinaison de sélecteur. Ce poids permet de savoir si un sélecteur va surcharger une règle déjà appliquée aux éléments correspondants à ce sélecteur. Le calcul du poids des sélecteurs à évoluer avec CSS3. Il a été simplifié et ne prend plus en charge le caractère *inline* des styles qui est de toute façon prioritaire (voir note en fin de paragraphe).

Ce poids est exprimé par trois nombres *a*, *b*, *c* où :

- *a* : nombre de sélecteurs d'identifiants (#).

- *b* : nombre de sélecteurs d'attributs de classes et de pseudo-classes.
- *c* : nombre de sélecteurs de balises et de pseudo-éléments.

Les sélecteurs dans la pseudo-classe de négation sont comptabilisés mais pas la pseudo-classe elle-même.

Cet exemple extrait de la spécification du W3C montre bien les différents poids des sélecteurs :

```
*                /* a=0 b=0 c=0 -> specificity =   0 */
li               /* a=0 b=0 c=1 -> specificity =   1 */
ul li            /* a=0 b=0 c=2 -> specificity =   2 */
ul ol+li         /* a=0 b=0 c=3 -> specificity =   3 */
h1 + *[rel=up]   /* a=0 b=1 c=1 -> specificity =  11 */
ul ol li.red     /* a=0 b=1 c=3 -> specificity =  13 */
li.red.level     /* a=0 b=2 c=1 -> specificity =  21 */
#x34y            /* a=1 b=0 c=0 -> specificity = 100 */
#s12:not(foo)    /* a=1 b=0 c=1 -> specificity = 101 */
```

On voit ici qu'un sélecteur plus précis sera prédominant sur un sélecteur plus général (ul li {} = 002 et ul ol+li {} = 003)

Note : il est possible de surcharger les règles CSS avec le mot-clé !important afin de prendre « l'avantage » sur les règles de l'auteur. Cette règle est surtout utilisée pour redéfinir l'aspect de fragments d'origines externes ayant des styles CSS sous forme d'attributs style="" et ayant donc un poids maximal.

2.3. Les @-règles

Les @-règles (*at-rules*) sont des règles qui étendent la simple syntaxe d'une déclaration de type sélecteur-bloc. Ces @-règles sont précédées de l'arobase (@) suivie d'un mot-clé. Il en existe deux types, des règles simples identiques à une déclaration classique et des règles identificateurs de bloc, entourées d'accolades ({ et }). Ces deuxièmes règles peuvent contenir des déclarations classiques de type propriété: valeur; ou des blocs de déclarations classiques de type sélecteur{propriété: valeur;}.

- @charset permet de définir l'encodage du fichier CSS.

```
@charset 'UTF-8';
```

- @import permet de charger une feuille de style externe.

```
@import url('style.css');
```

Note : lors de l'utilisation de ces règles, elles doivent être écrites en premier dans le CSS, même avant tout commentaire.

- @font-face permet de définir et de charger une police externe qui n'est pas disponible chez l'utilisateur. Son utilisation est détaillée dans le chapitre sur la typographie.
- @page définit un nouveau modèle de boîte : la page. Elle est utilisée pour les médias paginés comme l'impression par exemple.

```
@page{
    size: 16cm 20cm;
    margin: 1cm;
```

```
}
```

- @media permet de cibler un type de media (*screen*, *print*, *speech*...) afin d'appliquer des CSS alternatives. Son utilisation est détaillée dans le chapitre sur le mobile.

```
@media print{
	body{
		font-size: 12pt;
	}
}
```

CSS3 apporte un support de nouvelles @-règles comme par exemple @counter-style, @namespace, @viewport, @supports et développe l'utilisation de @media avec les *media-queries*.

2.4. Les listes

CSS3 apporte de nombreuses améliorations pour la prise en charge des styles de listes dans le HTML, notamment dans le cadre du contrôle du style des puces qui peuvent désormais être complètement redéfinies. Par rapport à CSS2.1, la propriété méconnue display: marker; a été remplacée par le pseudo-élément ::marker qui désigne le marqueur de liste, qu'elle soit à puce ou numérotée. Cette fonctionnalité n'est cependant pour l'instant supportée par aucun des navigateurs modernes.

Les styles de listes sont donc personnalisables grâce à la règle @counter-style décrite plus loin dans ce chapitre. Il est dès lors possible de définir l'aspect des « marqueurs » de liste selon la profondeur des éléments constituant les listes.

> Lors de l'écriture de cet ouvrage, il est question d'inclure dans la spécification le rendu des arborescences via les styles de listes.

Les balises concernées sont les balises dont la valeur de la propriété display est égal à list-item. C'est le cas pour la feuille de style par défaut des éléments li. Il est néanmoins possible d'appliquer ce display à n'importe quelle balise, cette dernière est alors à même d'accepter le pseudo-élément ::marker pour représenter son marqueur de liste.

Les propriétés list-style-type, list-style-image et list-style-position sont toujours d'actualité ainsi que le raccourci associé list-style dont voici un rappel :

```
ul li { list-style: outside url('mapuce.png') disc; }
```

Cet exemple défini une liste avec l'image *mapuce.png* comme puce ou un disque si l'image n'est pas disponible, le tout, en dehors du bloc représentant l'élément li.

2.4.1. Styles de listes

Les styles de listes prédéfinis en CSS2.1 sont peu nombreux, nous pouvons tout de même les citer :

- decimal, decimal-leading-zero
- lower-roman, upper-roman
- lower-greek
- lower-latin, upper-latin

- armenian, georgian
- lower-alpha, upper-alpha

Le module CSS3 *Lists and Counters* propose une @-règle pour définir le style des listes, la règle : @counter-style.

Cette règle n'est pour l'instant reconnue par aucun des navigateurs modernes mais espérons qu'elle le soit prochainement, car elle augure des listes surprenantes.

Voici rapidement sa syntaxe que nous ne détaillerons pas ici ainsi que quelques exemples de listes prédéfinies :

Une liste dont les puces sont composées de '✓'

```
@counter-style check{
	type: repeating;
	glyphs: '\2713';
	suffix: '';
}
```

Une liste numérique complexe basée sur les chiffres romains, valide pour les valeurs de 1 à 4999.

```
@counter-style upper-roman{
	type: additive;
	range: 1 4999;
	additive-glyphs: 1000 '\4D', 900 '\43\4D', 500 '\44', 400 '\43\44', 100 '\43', 90 '\58\43', 50 '\4C', 40 '\58\4C', 10 '\
}
```

> **Rappel** des valeurs de chiffres romains et des glyphes correspondants :
> 1000:M 900:CM 500:D 400:CD 100:C 90:XC 50:L 40:XL 10:X 9:IX 5:V 4:IV 1:I

Voici pour vous mettre l'eau à la bouche des listes basées sur des glyphes numériques :

- La liste à nombre cerclé : '❶' '❷' '❸' '❹' '❺' '❻' '❼' '❽' '❾' '❿'
- La liste 'dé': '⚀','⚁','⚂','⚃','⚄','⚅'

Ou bien des suites utilisant les glyphes d'autres alphabets :

- Enumération en Thaï : '๐' '๑' '๒' '๓' '๔' '๕' '๖' '๗' '๘' '๙'
- Alphabet hadiyya : 'ሀ' 'ለ' 'መ' 'ረ' 'ሰ' 'ሸ' 'ቀ' 'በ' 'ተ' 'ቸ' 'ነ' 'ኀ' 'ኘ' 'ከ' 'ጨ' 'ጰ' 'ፈ' 'ፐ'

Et bien d'autres encore.

2.5. Les compteurs

Les compteurs sont disponibles depuis CSS2.1 mais sont peu utilisés, nous allons voir ensemble qu'ils peuvent nous permettre de fabriquer une hiérarchie complète d'un document structuré.

Les compteurs sont basés sur l'utilisation conjointe de deux propriétés et d'une fonction (ou plutôt deux, car il y a une variante *cf.* compteur récursif). Les propriétés counter-reset et counter-increment permettent de définir les valeurs de départ et d'incrément des compteurs, la fonction counter() permet d'en afficher la valeur. Cependant, pour afficher cette valeur, il faudrait pouvoir modifier la puce (ou nombre) généré par la propriété display:list-item; ! Nous venons de voir que cette possibilité, offerte par le pseudo-élément ::marker n'est pas encore disponible sur les navigateurs. Nous allons donc détourner

l'utilisation du pseudo-élément ::before pour pallier ce manque.

Il est alors possible d'affecter nos compteurs à n'importe quel élément HTML. Pour illustrer ce fonctionnement, nous allons construire une hiérarchie de titre h1 et h2 que nous allons numéroter de façon automatique.

Tout d'abord, construisons une structure de titres nous permettant de bien visualiser les valeurs des compteurs :

```
<h1>Titre 1</h1>
	<h2>Titre 1.1</h2>
	<h2>Titre 1.2</h2>
	<h2>Titre 1.3</h2>
<h1>Titre 2</h1>
	<h2>Titre 2.1</h2>
	<h2>Titre 2.2</h2>
<h1>Titre 3</h1>
<h1>Titre 4</h1>
	<h2>Titre 4.1</h2>
	<h2>Titre 4.2</h2>
<h2>Titre 4.3</h2>
```

Nous allons ensuite définir deux compteurs, *T1* et *T2*, un pour chaque niveau de titre.

Nous initialisons nos compteurs à 0 grâce à la propriété counter-reset. La valeur 0 est ici facultative car c'est la valeur par défaut. Il est possible d'initialiser plusieurs compteurs à la fois. C'est même indispensable car la cascade ferait que la première règle serait ignorée si l'on écrivait les deux l'une à la suite de l'autre.

```
body{
	counter-reset: T1 0 T2 0;
}
```

Nous pouvons alors définir l'incrément du compteur *T1*, celui-ci doit augmenter à chaque h1 rencontré :

```
h1{
	counter-increment: T1;
}
```

Poursuivons avec le compteur *T2*, qui lui doit augmenter avec les h2 :

```
h2{
	counter-increment: T2;
}
```

Passons maintenant à l'affichage à l'aide du pseudo-élément ::before et de la propriété content. On insère donc avec la fonction counter() la valeur actuelle du compteur *T1* (figure 2.15, cas 1).

```
h1::before{
	content: counter(T1);
	color: darkorchid;
}
```

Optimisons avec une amélioration du premier compteur en lui concaténant une parenthèse et en

mettant en place le second compteur (figure 2.15, cas 2).

```
h1::before{
        content: counter(T1)') ';
        color: darkorchid;
}
h2::before{
        content: counter(T2)' ';
        color: orchid;
}
```

Ce n'est sûrement pas le résultat escompté !

Il faut effectivement remettre le compteur *T2* a 0 pour chaque h1 rencontré, la propriété counter-reset nous permet de faire cela. Profitons-en pour afficher une numérotation complète à l'aide des deux compteurs pour h2 (figure 2.15, cas 3).

```
h1{
        counter-increment: T1;
        counter-reset: T2;
}
...
h2::before{
        content: counter(T1)'.'counter(T2)' - ';
        color: orchid;
}
```

Fig. 2.15 Exemple de numérotation automatique totalement personnalisable

Counter(s)	8.0	2.0	4.0	3.1	9.0	3.2	10.0	2.1

Fig. 2.16 Tableau de compatibilité des compteurs

2.5.1. Aller plus loin : le compteur récursif

La fonction counters() est encore plus puissante car elle permet de s'affranchir de la profondeur de votre hiérarchie. Un seul compteur est donc suffisant pour numéroter une hiérarchie complète basée sur une balise unique. Cette fonction prendra toute sa force lors de l'implémentation du pseudo-élément ::marker, cette lacune nous obligeant encore une fois à détourner le pseudo-élément ::before. La

fonction counters() prend deux paramètres, le premier étant le nom du compteur, le second, le caractère séparateur qui va être intercalé entre chaque numéro.

Voici un exemple pour illustrer cette fonction : nous allons numéroter une liste à puce à plusieurs niveaux d'imbrication.

```
<ul>
	<li>Mammifères
		<ul>
			<li>Aquatiques
				<ul>
					<li>Ornithorynque</li>
					<li>Desman</li>
				</ul>
			</li>
			<li>Terrestres
				<ul>
					<li>Pangolin</li>
					<li>Echidné</li>
				</ul>
			</li>
		</ul>
	</li>
	<li>Oiseaux
		<ul>
			<li>Volants
				<ul>
					<li>Colibri</li>
				</ul>
			</li>
			<li>Terrestres
				<ul>
					<li>Kiwi</li>
					<li>Emeux</li>
					<li>Casoar</li>
				</ul>
			</li>
		</ul>
	</li>
	<li>Poissons
		<ul>
			<li>Blobfish</li>
			<li>Goliath Tigerfish</li>
		</ul>
	</li>
</ul>
```

Appliquons le compteur unique à cette liste en masquant les puces :

```
ul{
	counter-reset: item;
	list-style-type: none;
}
```

Puis affichons les valeurs du compteur, chaque étape étant séparée par un point, le tout suivi d'un espace :

```
ul li::before{
        counter-increment: item;
        content: counters(item,".")' ';
}
```

On obtient directement la figure 2.17.

1 Mammifères
1.1 Aquatiques
1.1.1 Ornithorynque
1.1.2 Desman
1.2 Terrestres
1.2.1 Pangolin
1.2.2 Echidné
2 Oiseaux
2.1 Volants
2.1.1 Colibri
2.2 Terrestres
2.2.1 Kiwi
2.2.2 Emeux
2.2.3 Casoar
3 Poissons
3.1 Blobfish
3.2 Goliath Tigerfish

Fig. 2.17 Résultat du compteur récursif

Les éléments avec display:none n'incrémentent pas les compteurs, ce qui n'est pas le cas avec ceux en visibility:hidden.

2.6. Les couleurs

Nous pourrions nous demander ce que CSS3 peut apporter de plus à la notion de couleur dans les pages web vu que la précédente version proposait déjà une palette de 65 millions de couleurs grâce à la notation hexadécimale #RRVVBB ou la fonction rgb(r,v,b). Sur ce point précis effectivement pas d'amélioration et il paraît évident qu'il n'est nul besoin d'étoffer cette palette étant donné que l'œil humain a une capacité de discernement des couleurs proches de 2 millions, un seuil bien en deçà de ce qu'il est possible de représenter.

C'est donc ailleurs qu'il faut chercher la nouveauté. Commençons par regarder les couleurs dont nous disposons.

2.6.1. Mots clés étendus

La spécification CSS2.1 ne fournit que 17 mots-clés (figure 2.18), raccourcis désignant les couleurs primaires RVB et leurs dérivés, plus l'orange, le blanc, le noir et deux niveaux de gris.

aqua #00FFFF	black #000000	blue #0000FF	fuchsia #FF00FF	gray #808080	green #008000	lime #00FF00
maroon #800000	navy #000080	olive #808000	orange #FFA500	purple #800080	red #FF0000	silver #C0C0C0
teal #008080	yellow #FFFF00	white #FFFFFF				

Fig. 2.18 Les couleurs nommées CSS 2.1
On remarque que certaines couleurs très connues
comme cyan ou magenta sont en fait des « alias » vers les couleurs de base.

On retrouve donc ces couleurs dans la liste des 147 couleurs nommées proposées par le W3C dont voici une liste compacte, donnée à titre indicatif (figure 2.19).

aliceblue #F0F8FF	antiquewhite #FAEBD7	aqua #00FFFF	aquamarine #7FFFD4	azure #F0FFFF	beige #F5F5DC	bisque #FFE4C4
black #000000	blanchedalmond #FFEBCD	blue #0000FF	blueviolet #8A2BE2	brown #A52A2A	burlywood #DEB887	cadetblue #5F9EA0
chartreuse #7FFF00	chocolate #D2691E	coral #FF7F50	cornflowerblue #6495ED	cornsilk #FFF8DC	crimson #DC143C	cyan #00FFFF
darkblue #00008B	darkcyan #008B8B	darkgoldenrod #B8860B	darkgray #A9A9A9	darkgreen #006400	darkgrey #A9A9A9	darkkhaki #BDB76B
darkmagenta #8B008B	darkolivegreen #556B2F	darkorange #FF8C00	darkorchid #9932CC	darkred #8B0000	darksalmon #E9967A	darkseagreen #8FBC8F
darkslateblue #483D8B	darkslategray #2F4F4F	darkslategrey #2F4F4F	darkturquoise #00CED1	darkviolet #9400D3	deeppink #FF1493	deepskyblue #00BFFF
dimgray #696969	dimgrey #696969	dodgerblue #1E90FF	firebrick #B22222	floralwhite #FFFAF0	forestgreen #228B22	fuchsia #FF00FF
gainsboro #DCDCDC	ghostwhite #F8F8FF	gold #FFD700	goldenrod #DAA520	gray #808080	green #008000	greenyellow #ADFF2F
grey #808080	honeydew #F0FFF0	hotpink #FF69B4	indianred #CD5C5C	indigo #4B0082	ivory #FFFFF0	khaki #F0E68C
lavender #E6E6FA	lavenderblush #FFF0F5	lawngreen #7CFC00	lemonchiffon #FFFACD	lightblue #ADD8E6	lightcoral #F08080	lightcyan #E0FFFF
lightgoldenrodyellow #FAFAD2	lightgray #D3D3D3	lightgreen #90EE90	lightgrey #D3D3D3	lightpink #FFB6C1	lightsalmon #FFA07A	lightseagreen #20B2AA
lightskyblue #87CEFA	lightslategray #778899	lightslategrey #778899	lightsteelblue #B0C4DE	lightyellow #FFFFE0	lime #00FF00	limegreen #32CD32
linen #FAF0E6	magenta #FF00FF	maroon #800000	mediumaquamarine #66CDAA	mediumblue #0000CD	mediumorchid #BA55D3	mediumpurple #9370DB
mediumseagreen #3CB371	mediumslateblue #7B68EE	mediumspringgreen #00FA9A	mediumturquoise #48D1CC	mediumvioletred #C71585	midnightblue #191970	mintcream #F5FFFA
mistyrose #FFE4E1	moccasin #FFE4B5	navajowhite #FFDEAD	navy #000080	oldlace #FDF5E6	olive #808000	olivedrab #6B8E23
orange #FFA500	orangered #FF4500	orchid #DA70D6	palegoldenrod #EEE8AA	palegreen #98FB98	paleturquoise #AFEEEE	palevioletred #DB7093
papayawhip #FFEFD5	peachpuff #FFDAB9	peru #CD853F	pink #FFC0CB	plum #DDA0DD	powderblue #B0E0E6	purple #800080
red #FF0000	rosybrown #BC8F8F	royalblue #4169E1	saddlebrown #8B4513	salmon #FA8072	sandybrown #F4A460	seagreen #2E8B57
seashell #FFF5EE	sienna #A0522D	silver #C0C0C0	skyblue #87CEEB	slateblue #6A5ACD	slategray #708090	slategrey #708090
snow #FFFAFA	springgreen #00FF7F	steelblue #4682B4	tan #D2B48C	teal #008080	thistle #D8BFD8	tomato #FF6347
turquoise #40E0D0	violet #EE82EE	wheat #F5DEB3	white #FFFFFF	whitesmoke #F5F5F5	yellow #FFFF00	yellowgreen #9ACD32

Fig. 2.19 Les couleurs nommées en CSS3

Le rendu écran sera probablement différent du rendu papier alors que le fait d'utiliser des couleurs RVB garantit un rendu identique sur des écrans calibrés. Il est donc judicieux d'utiliser des outils numériques de comparaisons de couleurs pour vérifier la cohérence de votre modèle/jeu de couleurs.

Il faut ajouter à cette liste la couleur transparent qui peut se résumer à la couleur noire à 0 % d'opacité. Nous verrons par la suite comment représenter cette couleur grâce aux fonctions.

On peut également noter l'ajout de la valeur currentColor qui représente la couleur de la propriété color ou la couleur héritée par la cascade lors de l'utilisation de la couleur courante dans d'autres règles.

2.6.2. Opacity

La grande nouveauté, même si les versions préfixées sont disponibles depuis quelques temps, c'est la gestion de l'opacité des éléments. Cette opacité peut s'effectuer de deux manières. Il est possible de

définir l'opacité d'un élément ou l'opacité d'une couleur. Commençons par voir ensemble la première solution, nous préciserons l'utilisation de la seconde un peu plus tard.

La propriété opacity permet donc de définir la transparence d'un élément, ce qui inclut l'arrière-plan, les bordures, le texte, les images et l'ensemble de ses enfants. Elle prend une valeur comprise entre 0.0 et 1.0, 0.0 représentant la transparence totale, 1.0 l'opacité totale.

Un titre avec l'opacité à 1 (figure 2.20)

```
<h1>CSS3 <span><img src="css3.png" alt="css3 Rock's" /></span></h1>
h1{
	padding: 10px;
	background: #e44d26;
	border: 10px solid #f16529;
	color: #fff;
	opacity: 1;
}
```

Fig. 2.20 L'opacité est à 1 : rien n'est transparent

Un titre avec l'opacité à 0.3 (figure 2.21)

```
h1{
	padding: 10px;
	background: #e44d26;
	border: 10px solid #f16529;
	color: #fff;
	opacity: 0.3;
}
```


Fig. 2.21 L'opacité est à 0.3 : l'élément est légèrement transparent

opacity	9.0	2.0	4.0	3.1	9.0	3.2	10.0	2.1

Fig. 2.22 Tableau de compatibilité d'opacity
Les navigateurs Internet Explorer jusqu'à la version 8 peuvent réaliser la transparence grâce à la notation propriétaire filter:alpha(opacity=35).

2.6.3. Les fonctions rgb/rgba et hsl/hsla

L'utilisation des couleurs en CSS peut être fait sous forme numérique, soit en utilisant la notation

hexadécimale, le signe # suivi des valeurs rouge puis vert puis bleu représentées par deux caractères hexadécimaux ou sa version courte dans le cas où les deux caractères sont égaux pour chaque couleur, ex. : #FF8833 peut s'écrire #F83.

Il est également possible de définir les couleurs avec la fonction rgb(rouge,vert,bleu) qui prend pour chaque paramètre soit la valeur de la couleur de 0 à 255 soit un pourcentage de 0 % à 100 %.

La fonction rgb() a été étendue pour permettre l'indication de l'opacité directement dans la couleur. La fonction devient alors rgba(rouge,vert,bleu,alpha) où *alpha* représente l'opacité de la couleur de 0.0 à 1.0 avec la même signification que la propriété opacity.

On notera cependant que cette opacité s'applique à la couleur uniquement à la différence de la propriété opacity qui s'applique à tout l'objet.

Un titre avec une opacité à 0.3 sur l'arrière-plan (figure 2.23)

```
h1{
	padding: 10px;
	background: rgba(228,77,38,0.3);
	border: 10px solid #f16529;
	color: #fff;
}
```

Fig. 2.23 L'opacité de l'arrière-plan est à 0,3

On voit que le texte, l'image et la bordure ne sont pas impliqués par cette notion de transparence.

L'utilisation du modèle RVB, s'il est bien adapté au support écran qui est basé dessus, n'est pas facile à appréhender sans l'aide d'un convertisseur. Il est effectivement peut intuitif de faire varier l'ensemble des paramètres d'une couleur, simplement pour la rendre un peu plus claire. C'est pourquoi CSS3 introduit l'utilisation du modèle HSL (*Hue*, *Saturation*, *Lightness* ou Teinte, Saturation, Luminance en français) pour définir les couleurs. Ces fonctions convertissent bien entendu la couleur au modèle RVB de façon transparente pour nous.

Le modèle HSL peut se résumer ainsi :

- La **teinte** (*Hue*) représente l'angle de 0 à 360° (0° = 360° = rouge) indiquant la couleur sur le cercle chromatique représentant les couleurs de l'arc en ciel.
- La **saturation** (*Saturation*) indique le degré d'intensité de la couleur. Une saturation de 0 % correspondant au niveau de gris précisé par la luminance.
- La **luminance** (*Lightness*) correspond à l'éclat de la couleur, ce qui revient à l'ajout de blanc ou de noir dans la teinte. À 50 % de luminance, la teinte n'est pas altérée. À 100 %, la teinte est blanche et à 0 % la teinte est noire.

Le modèle HSL se représente parfois comme un cercle chromatique avec les couleurs de l'arc en ciel qui définissent la teinte. Pour chaque teinte, un triangle est symbolisé, avec la hauteur qui correspond à la saturation et la base qui correspond à la luminance. Une couleur avec une luminance à 0 % est toujours égale à noir et une couleur avec une luminance à 100 % est toujours égale à blanc

(figure 2.24).

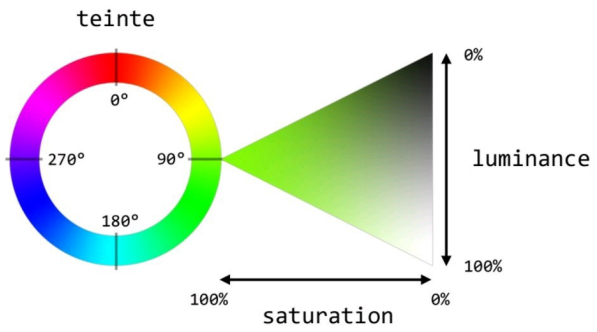


Fig. 2.24 Modèle HSL, la valeur de la couleur à la pointe du triangle est Teinte : 90%, Saturation : 100 %, Luminance : 50 %.

Deux fonctions sont donc disponibles pour définir les couleurs selon ce modèle, hsl(hue, saturation, lightness) et hsla(hue, saturation, lightness, alpha). La valeur Hue variant de 0 à 360, Saturation et Lightness de 0% à 100% et Alpha de 0.0 à 1.0.

L'écriture de la couleur #99CC33:

```
color: hsl( 80, 60%, 50%);
```

équivaut à

```
color: rgb( 153, 204, 51);
```

L'interêt d'utiliser le modèle HSL en lieu et place du modèle RVB, est que la notation hsl/hsla nous permet de choisir une teinte plus sombre ou plus claire par rapport à la teinte de base, et cela avec une facilité déconcertante. Il nous suffit de modifier la luminance (figure 2.25).

Teinte plus claire

```
color: hsl( 80, 60%, 60%);
```

Teinte plus foncée

```
color: hsl( 80, 60%, 40%);
```

Fig. 2.25 Choix de trois teintes de couleurs

Pour pouvoir reproduire cela avec la notation RGB, il nous aurait fallu utiliser un logiciel pour trouver cette valeur. Il existe des tables de conversions HSL <> RVB mais cela n'a pas vraiment d'intérêt pour nous. Attention toutefois, le modèle HSL (TSL) de Photoshop et d'autres logiciels graphiques ne correspond pas à celui de CSS en ce qui concerne la lumière.

Note : il existe de nombreux sites qui nous assiste dans le choix de nos couleurs, que ce soit pour définir des palettes de couleurs monochromes, analogues, complémentaires, complémentaires adjacentes ou tout simplement primaires ou secondaires. Le plus connu d'entre eux, kuler édité par Adobe, nous facilite grandement le travail (figure 2.26).

Kuler
URL : http://kuler.adobe.com.
Citons également le Chromafiltre du site pourpre.com.
URL : http://pourpre.com/outils/chromafiltre.php

Fig. 2.26 Kuler

rgba, hsl, hsla	9.0	3.0	4.0	3.1	10.0	3.2	10.0	2.1

Fig. 2.27 Tableau de compatibilité de rgba(), hsl() et hsla()

2.7. Les unités

CSS3 a vu naître son lot de nouvelles unités notamment par l'ajout de nouveaux modules et donc de nouvelles fonctionnalités (l'unité secondes pour les animations par exemple). Certaines d'entre elles étaient attendues par la communauté mais globalement il n'y a pas beaucoup de changement de ce côté-là.

2.7.1. Les unités relatives

Les unités relatives sont appelées ainsi car elles ne représentent pas une longueur prédéfinie mais une longueur relative à une référence. Les nouvelles unités présentées en fin de tableau ne sont pas toutes intégrées dans l'ensemble des moteurs, mais les exemples dans les chapitres concernés vous détailleront les cas d'utilisation de ces dernières (tableau 2.15).

Tab. 2.15

Unité	Définition
%	Représente une valeur en pourcentage. La référence peut être le parent, l'élément lui même ou tout autre valeur.
em	Représente une proportion de la taille de la police courante.
ex	Représente la hauteur du glyphe 'x' (ou équivalent) dans la police courante
ch	Représente la dimension du glyphe '0'
rem	Représente la taille de la police de l'élément html.
fr	Représente une fraction de l'espace restant.
dpi	Représente la résolution du périphérique.
vw	Représente la largeur du viewport du périphérique.
vh	Représente la hauteur du viewport.
vm	Représente la plus petite valeur du viewport (largeur ou hauteur).

2.7.2. Les unités absolues

Ces unités sont applicables sur toutes les propriétés nécessitant une valeur correspondant à une distance. Elles sont cependant totalement dépendantes de la résolution du média sur lequel on visualise la page, il faudra donc être vigilant quant à l'utilisation des unités absolues (tableau 2.16).

Tab. 2.16

Unité	Définition
cm	centimètre
mm	millimètre
in	pouce, c'est environ 2,54 cm
px	pixel, initialement 1 px est égal à 1/96 ème de pouce
pt	point, initialement 1 pt est égal à 1/72ème de pouce
pc	pica, 1 pc est égal 12 pt

2.7.3. Les unités d'angles

Nous les utiliserons notamment pour définir des transformations. Notons la présence de l'unité turn, très utile pour définir des cycles de rotations, son implémentation est bientôt complète, Mozilla l'ayant ajoutée à Firefox 13 (tableau 2.17).

Tab. 2.17

Unité	Définition
deg	degré, 1 tour = 360 deg
grad	grade, 1 tour = 400 grad
rad	radian, 1 tour = 2π = 6,28 rad
turn	tour, 0,5 tour = 180 deg

2.7.4. Les unités de temps

Nous les utiliserons pour les transitions et les animations (tableau 2.18) mais sachez qu'elles sont aussi utiles pour les media de type audio (media speech).

Tab. 2.18

Unité	Définition
s	seconde
ms	milliseconde, 1 000 ms = 1s

2.8. Les fonctions

L'utilisation des fonctions n'est pas nouvelle dans CSS, en effet vous connaissez déjà sûrement la fonction url() qui permet de charger le contenu de l'adresse HTTP passée en paramètre ou encore rgb() qui permet de définir une couleur. CSS3 introduit de nouvelles fonctions qui permettent de réaliser des opérations qui sortent du cadre habituel de CSS.

2.8.1. calc

La fonction calc() permet de calculer une valeur en effectuant des opérations arithmétiques simples, addition, soustraction, multiplication, division. Elle permet donc par exemple de calculer une largeur dynamique tout en ménageant une marge constante : width : calc(100% - 50px);. Elle peut aussi servir à faire des calculs plus complexes pour prendre en compte des éléments multiples : margin-left: calc(20%/3 - 2em);.

Note : les fonctions min() et max() ont été supprimées de la spécification.

Malheureusement, même si elle est alléchante, cette fonction est encore très peu implémentée. La version préfixée est de rigueur et le résultat n'est pas toujours celui escompté. Mozilla propose tout de même une version complète avec notamment l'utilisation de l'opérateur de modulo (%). Les autres navigateurs devraient probablement suivre la marche, c'est d'ailleurs en bonne voie dans Webkit.

calc()	9.0	4.0 -moz-	19.0 -webkit-					

Fig. 2.28 Tableau de compatibilité de calc()

2.8.2. attr

La fonction attr() n'est pas nouvelle, elle existait déjà dans CSS2.1 mais son action se limitait à la valeur de la propriété content et aux valeurs des attributs correspondant au sélecteur courant et de type chaîne de caractère (figure 2.29). Avec la spécification CSS3, elle peut renvoyer n'importe quel type de valeur, accompagné d'un paramètre spécifiant ce type et éventuellement d'une valeur par défaut. Elle n'est plus limitée à la propriété content, on peut donc l'appliquer à n'importe quelle propriété compatible avec la valeur récupérée. Tout cela est très prometteur mais malheureusement, pour l'instant, son implémentation n'a pas évoluée est donc toujours limitée à son ancienne utilisation.

Une petite évolution notable est amenée par le HTML5 et les attributs data-* (*custom data attributes*). Cette nouvelle forme d'attributs HTML permet de définir des attributs personnalisés qui n'ont de sémantique que pour son auteur (et ne sont pas voués à en avoir ailleurs), ils sont donc totalement libres. Même s'ils sont surtout intéressants en JavaScript pour définir des métadonnées spécifiques à un élément, il est tout de même possible de définir les data-* attributes pour vos balises HTML et d'utiliser leurs valeurs dans la propriété content des pseudo-éléments ::before et ::after desdites balises.

```
width: attr( data-largeur, px, 200);
```

attribut | unité | défaut

Fig. 2.29 Syntaxe complète de attr()

Voici un exemple simplifié tiré du site de démo permettant l'affichage d'une donnée attributaire utilisable dans tous les principaux navigateurs modernes (figure 2.30) :

Liste HTML

```
<ul id="chart">
	<li data-population="20000">1990</li>
	<li data-population="50000">2000</li>
	<li data-population="70000">2010</li>
	<li data-population="120000">2020</li>
</ul>
```

La partie CSS se limite ici à l'affichage des données attributaires avec ::after.

Récupération de l'attribut en CSS

```
#chart li::after{
	content: attr(data-population);
	display: block;
```

```
        margin-top: -3em;
        background: rgba( 0, 0, 0, 0.6);
        color: #fff;
}
```

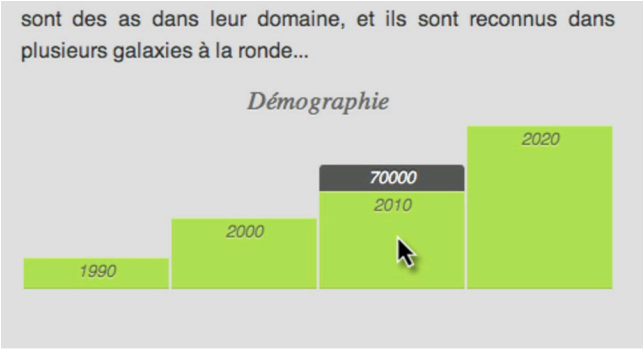


Fig. 2.30 Résultat sur le site support

Dans la spécification, l'écriture attr(population) sans la partie data- sera privilégiée. De plus avec une implémentation complète de la spécification, il aurait été possible de dimensionner le graphique directement à partir des données attributaires en écrivant une règle comme celle-ci :

```
#chart li {
        ...
        height: attr( calc( population/1000 ) , px );
        ...
}
```

2.9. Les curseurs

CSS2.1 introduisait la possibilité d'utiliser des curseurs personnalisés, CSS3 apporte de nouveaux curseurs standards qu'il est intéressant de montrer ici pour enrichir vos applications web (figure 2.31). Les curseurs personnalisés s'étoffent d'une coordonnée x et y permettant de décaler l'icône afin de la faire correspondre précisément avec le point utile du curseur.

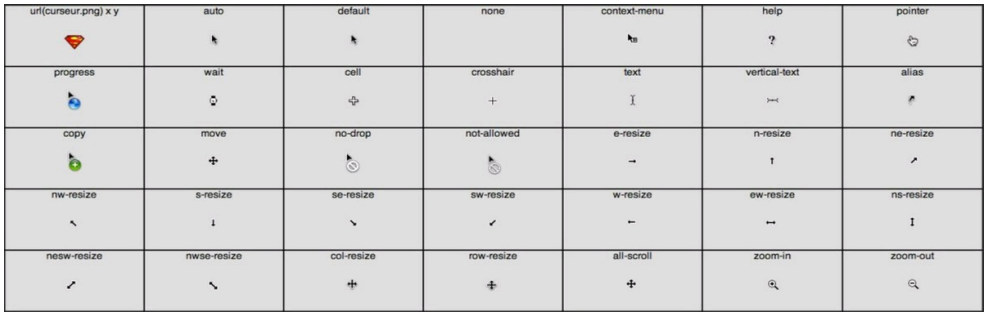


Fig. 2.31 Les curseurs CSS dans l'environnement Mac OS

2.10. Les éléments d'interfaces

CSS a toujours été un langage pauvre au niveau du formatage des formulaires, là ou d'autres langages proposent des mécanismes de *look&feel* pour changer l'aspect général des IHM (Interface Homme-Machine). Le choix a été fait de laisser le navigateur utiliser soit la représentation graphique du système d'exploitation soit une représentation interne du navigateur, sans laisser la possibilité au

développeur de la modifier. Une tentative a pourtant été faite pour introduire des propriétés CSS permettant de forcer un style d'IHM propre au système afin d'avoir une page conforme à l'aspect de l'environnement d'exécution du navigateur. Cette partie a aujourd'hui été supprimée de la spécification.

CSS3 n'offre pas beaucoup plus de possibilités. Parmi les options disponibles, citons les pseudo-classes :enabled, :disabled, :checked ou :indeterminate qui permettent de cibler les différents états des formulaires afin de les mettre en forme. Malheureusement, les possibilités offertes sont toujours aussi limitées.

Il reste néanmoins quelques libertés sur certains éléments :

- Les input de type submit, password, text et ses dérivés HTML5 (email, tel, search, etc.), textarea, button reconnaissent la majeure partie des propriétés CSS y compris dans les différents états grâce aux pseudo-classes citées ci-dessus.
- Les input de type radio, checkbox, select/option et la plupart des éléments HTML5 (range, date et dérivés, datalist, progress, etc.) offrent très peu de possibilités de modification, voire aucune.

HTML5 apporte la possibilité de forcer ou de valider la saisie de certains champs de formulaires. Cette caractéristique est accessible en CSS avec les nouvelles pseudo-classes suivantes et permet d'appliquer des styles en conséquence :

- :valid / :invalid : lors de la saisie d'un email invalide par exemple.
- :in-range / :out-of-range : lors de la saisie d'un nombre dépassant les bornes prévues.
- :required / :optional : lors de l'oubli d'un champ obligatoire.

Attention toutefois, le support des navigateurs est très hétérogène et il est difficile d'obtenir un rendu identique (figure 2.32).

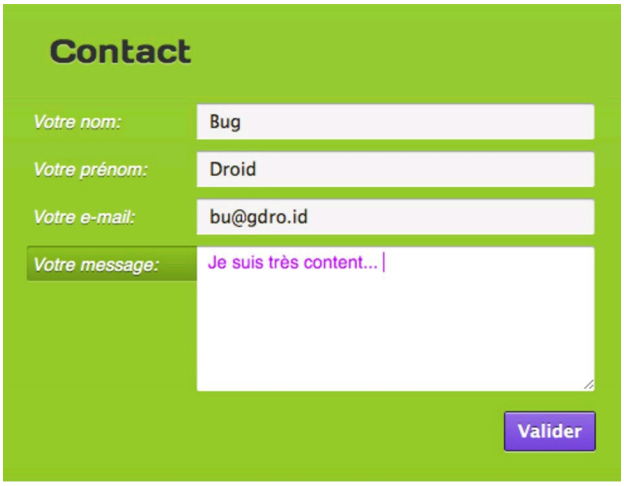


Fig. 2.32 Formulaire stylé en CSS

Notons tout de même l'ajout de la propriété resize, qui permet à l'utilisateur de modifier la taille d'un élément de la page. Un élément devient redimensionnable si sa propriété overflow est différente de visible et que la propriété resize est différente de none. Une « poignée » est alors symbolisée en bas à droite de l'élément et un curseur adapté est affiché. L'élément textarea a cette particularité par défaut.

Les valeurs possibles pour resize sont horizontal, vertical, both (les deux) ou none. Voici un exemple ou une div devient redimensionnable en hauteur (figure 2.33).

```
div{
```

```
        overflow: hidden;
        resize: vertical;
}
```

Fig. 2.33 Une poignée apparait pour agrandir l'élément

resize		4.0	4.0	4.0				

Fig. 2.34 Tableau de compatibilité de resize

Ce qu'il faut retenir...

À travers ce chapitre, nous avons fait le tour des éléments du langage CSS et plus particulièrement des nouveautés apportées par CSS3. Certaines sont plus utiles que d'autres mais toutes ont leur raison de se trouver dans la spécification et auront certainement leur heure de gloire dans une page sur la toile.

Nous avons détaillé le module Selectors CSS3 qui est, rappelons-le, l'un des premiers modules à avoir atteint le statut de recommandation. Nous avons également survolé les fonctions de couleurs et de contenu, qui étaient très attendues par la communauté même si calc() et attr() n'ont pas encore une maturité suffisante pour être utilisée dès aujourd'hui.

Nous pouvons donc poursuivre sur les chapitres suivants ou nous aborderons ces différentes propriétés aux travers d'exemples appliqués.

Les effets graphiques

Objectifs

En matière de design web, le « faire beau » a une importance primordiale. L'utilisation d'un logiciel graphique, tel que Photoshop, est alors indispensable pour créer des images qui seront ensuite appliquées à l'aide de la propriété background-image. Ces logiciels nous permettent de réaliser des ombres, des coins arrondis, des dégradés, ou des effets plus poussés comme des flous, des filtres de netteté ou encore de régler la luminosité ou le contraste d'une image.

CSS3 marque un tournant dans la création graphique pour le web, puisque les possibilités offertes au sein de la spécification y prennent une part importante. Et comme tout va pour le mieux dans le meilleur des mondes, la majorité de ces propriétés sont d'ores et déjà utilisables dans les navigateurs actuels.

L'objectif de ce chapitre est de vous présenter les différentes possibilités graphiques qu'offre CSS3.

3.1. Le graphisme avant CSS3

Avant de rentrer dans le vif du sujet, revenons quelques années en arrière.

Historiquement, la majeure partie des effets graphiques pour le web était réalisée à l'aide d'images. Par exemple, pour la création d'ombres autour d'un bloc ou pour la création de coins arrondis, plusieurs images étaient créées puis appliquées en CSS à l'aide de la règle background-image. Malheureusement, comme il n'est pas possible d'appliquer plusieurs images à l'intérieur d'un même élément (du moins en CSS2.1), il fallait alors recourir à plusieurs blocs (figure 3.1).

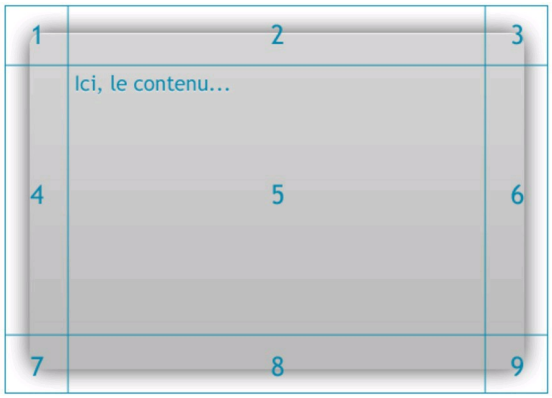


Fig. 3.1 La découpe d'une boîte pour les ombres et les coins arrondis (ancienne méthode)

Cela pose plusieurs problèmes :

- plusieurs éléments HTML doivent être créés, ce qui augmente le nombre de balises et leur gestion ;
- ces éléments doivent être positionnés correctement, ce qui engendre souvent des positionnements flottants et des décalages ;

- les images doivent être créées avec un logiciel graphique et tout le monde n'a pas les compétences requises ;
- les images doivent être « étirables » pour qu'elles puissent s'adapter à différentes largeurs et hauteurs possibles. Il faut donc prévoir des images répétables ;
- un plus grand nombre d'images implique plus de requêtes HTTP, ce qui ralentit forcément le chargement de la page ;
- une pixellisation des images est à prévoir si l'utilisateur agrandit la page ou si les images sont utilisées à des résolutions différentes (écran de bureau vs. écran mobile) ;
- la mise à jour est bien entendu plus complexe.

C'est pour toutes ces raisons que CSS3 tente de répondre aux désirs des graphistes en proposant des règles spécifiques qui permettent de s'affranchir de l'utilisation de logiciel pour certains effets simples. C'est le cas de la transparence, des ombres, des coins arrondis, des dégradés. Détaillons ensemble ces fonctionnalités.

3.2. La transparence

Commençons avec la gestion de la transparence. Son adoption sur le web a longtemps été retardée, notamment en raison du fait que la norme CSS2.1 ne fait état d'aucune règle de gestion de la transparence. C'est désormais chose faite avec CSS3 qui définit la règle opacity et les fonctions de couleurs rgba et hsla. Le mot-clé transparent est, quant à lui, étendu à une utilisation pour toutes les propriétés de couleur.

3.2.1. opacity

La règle opacity nous permet de jouer sur l'opacité d'un élément. Une valeur de 0 rend l'élément totalement transparent, alors qu'une valeur de 1 le rend totalement opaque. Voici un exemple d'opacité à 60 % appliqué sur une div :

```
div{
	opacity: 0.6;
}
```

Cette propriété a été ajoutée au sein de CSS3 en mars 2001. Depuis, son support a évolué et opacity est désormais présente dans la majorité des navigateurs. Internet Explorer offre le support depuis la version 9.

3.2.2. rgba, hsla, transparent

Contrairement à opacity qui joue sur l'opacité d'un élément, rgba, hsla et transparent permettent de jouer sur la transparence des couleurs appliquées. Cela permet par exemple de créer un fond transparent tout en gardant les textes opaques ou d'avoir des transparences différentes entre chaque élément d'un même parent. Voici un exemple d'utilisation de rgba et hsla en arrière-plan et sur les textes :

```
div{
	background: hsla(210, 100%, 50%, 0.3);
	color: rgba(0, 0, 0, 0.5);
}
```

Rappel : le chapitre précédent détaille ces notions.

3.3. Les ombres

La création d'ombres est une pratique courante qui permet de donner du relief ou de la profondeur à un graphisme. Les ombres permettent de simuler un éclairage afin de donner une certaine illusion du « vrai ». Sans ombres, tout paraît plat.

Sur le web, les ombres sont très souvent utilisées, notamment sur les boîtes. Cela permet, entre autre, de pouvoir créer une sorte de profondeur entre les différents éléments (figure 3.2).

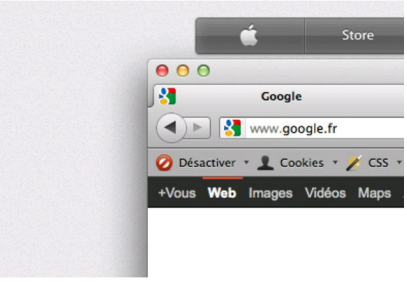


Fig. 3.2 Une ombre sur une fenêtre dans MacOS

Depuis peu, la création des ombres a été simplifiée par l'ajout de règles CSS dédiées. Il est à présent possible d'ajouter une ombre sur une boîte ou sur un texte, grâce aux propriétés box-shadow et text-shadow (*shadow* voulant dire ombre en français).

3.3.1. text-shadow

La propriété text-shadow ajoute une ou plusieurs ombres sur un texte. Une ombre est créée à partir de quatre paramètres (trois longueurs et une couleur) où les valeurs correspondent à (figure 3.3) :

- **décalage horizontal** (obligatoire)
  - une valeur positive décale l'ombre vers la droite,
  - une valeur négative décale l'ombre vers la gauche.
- **décalage vertical** (obligatoire)
  - une valeur positive décale l'ombre vers le bas,
  - une valeur négative décale l'ombre vers le haut.
- **valeur du flou** (optionnel)
  - une valeur de 0 donne une ombre nette,
  - une valeur positive provoque un flou gaussien,
  - les valeurs négatives sont interdites.
- **couleur (optionnel-obligatoire)**
  - toutes les définitions de couleurs sont autorisées (mot-clé, hexadécimal, fonctions avec ou sans alpha),
  - la couleur peut aussi être écrite avant les trois autres paramètres.

```
text-shadow: 2px 2px  2px  black;
```

décalage flou couleur

Fig. 3.3 Syntaxe de la propriété text-shadow

Les valeurs de décalage de l'ombre sont obligatoires, tandis que les valeurs de flou et de couleur sont optionnelles. Si ces valeurs ne sont pas précisées, la valeur de flou est de 0 et celle de la couleur est celle utilisée actuellement par le texte, définie par la règle color.

Dans les faits et notamment sur Chrome et Safari, si la couleur des textes n'est pas précisée, aucune ombre n'est créée.

Effet d'incrustation du texte (figure 3.4)

```
p{
	text-shadow: 0 1px 0 rgba( 255, 255, 255, 0.6);
}
```

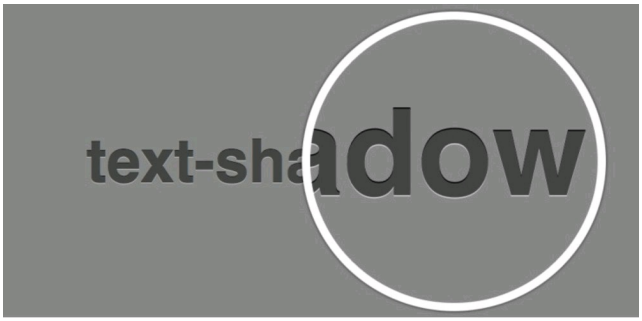


Fig. 3.4 Effet de text-shadow

Effet d'élévation du texte (figure 3.5)

```
p{
	text-shadow:2px 2px 2px rgba( 0, 0, 0, 0.6);
}
```

Fig. 3.5 Effet de text-shadow

Il est également possible d'ajouter plusieurs ombres simultanément en les séparant par une virgule. La première ombre s'applique alors juste sous le texte et les autres sont rendues en dessous, sans jamais superposer le texte.

Ombres multiples: effet de relief (figure 3.6)

```
p{
```

```
    text-shadow: 1px 1px 0 #333,
                 2px 2px 0 #333,
                 3px 3px 0 #333,
                 4px 4px 0 #333,
                 5px 5px 0 #333,
                 5px 5px 5px #333;
}
```

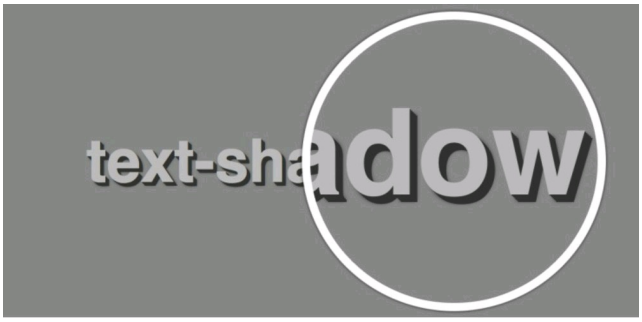


Fig. 3.6 Effet de text-shadow

text-shadow	10.0	3.5	4.0	4.0	9.5	3.2	10.0	2.1

Fig. 3.7 Tableau de compatibilité de text-shadow.
text-shadow peut être simulée avec des filtres propriétaires
pour les anciennes versions d'IE.

3.3.2. box-shadow

La propriété box-shadow ajoute une ou plusieurs ombres sur une boîte. Une ombre est créée à partir de six paramètres (quatre longueurs, une couleur et un mot-clé) où les valeurs correspondent à (figure 3.8) :

- **décalage horizontal** (obligatoire)
  - une valeur positive décale l'ombre vers la droite,
  - une valeur négative décale l'ombre vers la gauche.
- **décalage vertical** (obligatoire)
  - une valeur positive décale l'ombre vers le bas,
  - une valeur négative décale l'ombre vers le haut.
- **valeur du flou** (optionnel)
  - une valeur de 0 donne une ombre nette,
  - une valeur positive provoque un flou gaussien,
  - les valeurs négatives sont interdites.
- **valeur d'étendue** (optionnel)
  - une valeur positive augmente la taille de l'ombre, par rapport à celle de la boîte,
  - une valeur négative diminue la taille de l'ombre.
- **couleur** (optionnel-obligatoire)
  - toutes les définitions de couleurs sont autorisées (mot-clé, hexadécimal, fonctions avec ou

sans alpha),
    - la couleur peut aussi être écrite avant les quatre autres paramètres.
- **mot-clé**inset (optionnel)
    - crée une ombre interne, c'est-à-dire, à l'intérieur de la boîte.

```
box-shadow: 2px 2px  2px  black  inset;
```

décalage | flou | couleur | interne

Fig. 3.8 Syntaxe de la propriété box-shadow

Les valeurs de décalage de l'ombre sont obligatoires, tandis que les valeurs de flou, d'étendue, la couleur et le mot-clé inset sont optionnels. Si ces valeurs ne sont pas précisées, la valeur de flou et celle d'étendue sont de 0, la couleur est définie par le navigateur et l'ombre n'est pas interne.

Dans les faits, notamment sur Chrome et Safari, si la couleur n'est pas précisée, aucune ombre n'est créée sur la boîte. Il est donc préférable d'ajouter une couleur systématiquement.

Effet d'élévation d'une boîte (figure 3.9)

```
p{
	box-shadow: 0 0 8px black;
}
```

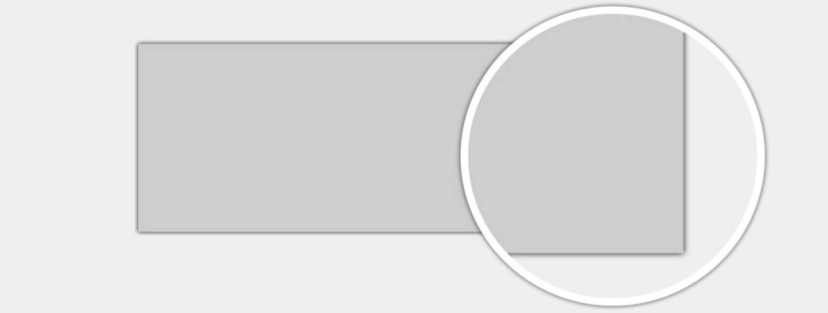


Fig. 3.9 Effet de box-shadow

Il est également possible d'ajouter plusieurs ombres simultanément, en les séparant par une virgule. Les ombres s'appliquent alors de haut en bas, sans jamais superposer la boite.

Effet d'incrustation d'une boîte (figure 3.10)

```
p{
	box-shadow: 0 -1px 0 white inset, 0 1px 8px black inset;
}
```

Fig. 3.10 Effet de box-shadow

Dernière possibilité, modifier l'étendue de l'ombre pour créer des ombres plus petites que l'élément.

Effet 3D sur une boîte (figure 3.11)

```
p{
	box-shadow: 0 1px 0 0 #999,
		0 3px 0 -1px #929292,
		0 5px 0 -2px #878787,
		0 7px 0 -3px #7B7B7B,
		0 9px 0 -4px #6E6E6E,
		0 11px 0 -5px #626262,
		0 13px 0 -6px #575757,
		0 6px 10px rgba(0,0,0,0.2);
}
```

Fig. 3.11 Effet de box-shadow

Normalement, l'ombre ajoutée ne modifie pas la taille de l'élément. Elle est dessinée à l'extérieur de la boîte. De plus, les ombres sont découpées au niveau des bordures, ce qui signifie qu'une boîte transparente ne laisse jamais apparaître l'ombre. Celle-ci paraît rognée au niveau de la boîte. Cela permet de pouvoir utiliser une couleur d'arrière-plan semi-transparente, sans voir l'ombre au travers.

box-shadow	9.0	3.5 -moz-	4.0	4.0 -webkit-	10.0	3.1 -webkit-	5.1	10.5	3.2 -webkit-	5.0	11.0	2.1 -webkit-

Fig. 3.12 Tableau de compatibilité de box-shadow.
box-shadow peut être simulée avec des filtres propriétaires
pour les anciennes versions d'IE.

3.3.3. Dégradation gracieuse

Contrairement à text-shadow qui n'est jamais préfixé, box-shadow nécessite l'utilisation des préfixes pour fonctionner dans la majorité des navigateurs. Pour box-shadow, il faudra ajouter -webkit- et -moz-.

Ajout des préfixes pour un fonctionnement dans la majorité des navigateurs

```
div{
        -webkit-box-shadow: 0 0 5px 3px black;
           -moz-box-shadow: 0 0 5px 3px black;
                box-shadow: 0 0 5px 3px black;
}
```

L'ajout des préfixes pour box-shadow doit être soumis à réflexion. En effet, la majorité des navigateurs récents supportent la règle non préfixée (*cf.* tableau de compatibilité). Ajouter les préfixes alourdit le code, et il n'est certainement plus utile de maintenir une telle syntaxe. Une ombre n'est qu'un élément de décor. Attention quand même au cas où le texte qui reçoit une ombre est de la même couleur que le fond, il ne sera plus lisible sans ombre. Une solution doit alors être mise en place.

3.4. Les coins arrondis

À l'instar des ombres, la création de coins arrondis est une pratique courante qui permet d'adoucir un élément de design. Sur une page web, les coins arrondis permettent d'alléger le rendu visuel des blocs, en lieu et place des coins carrés, nets.

Voici un exemple d'utilisation des coins arrondis sur le site twitter.com, pour le champ de recherche, le bouton d'écriture de nouveau *tweet*, ou pour le menu de profil (figure 3.13).

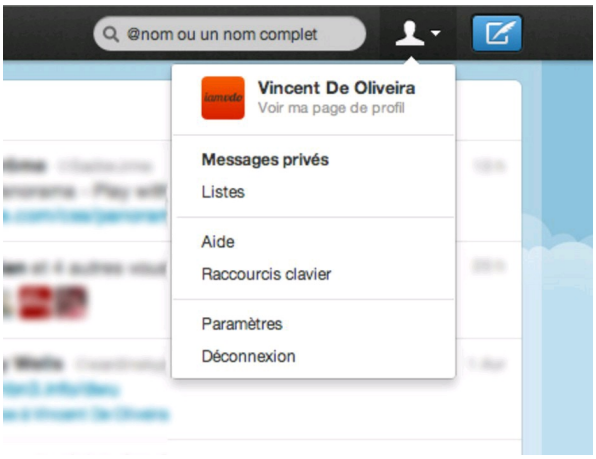


Fig. 3.13 Utilisation des coins arrondis sur twitter.com

La création de ces coins arrondis passe encore très souvent par la technique utilisée pour la création des ombres, à savoir, la création d'images de fond que nous appliquons sur différents blocs. Nous vous renvoyons vers l'explication complète plus haut.

Cette technique est devenue obsolète depuis l'apparition de la propriété border-radius qui permet d'arrondir les coins d'une boîte beaucoup plus facilement et proprement.

3.4.1. border-radius

La propriété border-radius permet d'ajouter des coins arrondis sur une boîte, en spécifiant une taille de rayon pour ses angles.

Comme pour l'utilisation des propriétés de marges (margin et padding), border-radius accepte une, deux, trois ou quatre valeurs séparées par des espaces (figure 3.14) :

- **une valeur** : les quatre arrondis sont identiques ;
- **deux valeurs** :
  - la première définit l'angle en haut à gauche et l'angle en bas à droite ;
  - la deuxième définit l'angle en haut à droite et l'angle en bas à gauche.
- **trois valeurs** :
  - la première définit l'angle en haut à gauche ;
  - la deuxième définit l'angle en haut à droite et l'angle en bas à gauche ;
  - la troisième définit l'angle en bas à droite.
- **quatre valeurs** :
  - les angles sont définis chacun leur tour, à partir de l'angle en haut à gauche puis en tournant dans le sens des aiguilles d'une montre.

```
border-radius: 20px 10px 0 20px;
```

rayons

Fig. 3.14 Syntaxe de la propriété border-radius

Les 4 coins sont arrondis de 20px (figure 3.15)

```
div{
	border-radius: 20px;
}
```

Fig. 3.15 Effet de border-radius

Chaque coin à son propre rayon (figure 3.16)

```
div{
	border-radius: 20px 50px 0 10px;
}
```

Fig. 3.16 Effet de border-radius

Il existe également des propriétés longues pour chaque coin, qui permettent de définir chaque coin indépendamment, comme ceci :

- border-top-left-radius : pour le coin supérieur gauche ;
- border-top-right-radius : pour le coin supérieur droit ;
- border-bottom-left-radius : pour le coin inférieur gauche ;
- border-bottom-right-radius : pour le coin inférieur droit.

Deux valeurs pour un même angle

Dans de plus rares cas, il est aussi possible de spécifier deux valeurs différentes pour chaque angle. Cela revient à définir une courbe, qui ne suit plus un cercle, mais une ellipse. Pour cela, il faut séparer les valeurs avec le caractère slash (/) :

Valeurs différentes pour un même angle (figure 3.17)

```
div{
	border-radius: 50px / 100px;
}
```

Fig. 3.17 Effet de border-radius avec deux valeurs

De plus, la syntaxe raccourcie marche également pour spécifier deux valeurs différentes pour chacun des quatre coins :

```
div{
	border-radius: 30px 20px 10px 0 / 10px 20px 30px 0;
}
```

Border-radius : 50 %

Si une valeur en pourcentage est appliquée pour border-radius, cette valeur se réfère à la largeur et à la hauteur de la boîte elle-même (contrairement aux règles CSS habituelles où une valeur en pourcentage se réfère souvent à la taille de son parent). Les marges internes (padding) et les bordures sont incluses dans cette taille. C'est pourquoi le fait de spécifier border-radius: 50% sur un élément lui ajoute des coins arrondis de sa demi-taille, et donc a pour effet de créer une ellipse (ou un rond si la boîte est carrée).

Un rond en CSS (figure 3.18)

```
div{
	width: 100px;
	height: 100px;
	padding: 25px;
```

```
        border: 5px solid #000;
        background: red;
            border-radius: 50%;
}
```

Fig. 3.18 Un rond en CSS

Attention, l'utilisation des pourcentages n'est pas supportée dans certaines anciennes versions de navigateurs.

border-radius	9.0	2.0 -moz-	4.0	4.0 -webkit-	5.0	3.1 -webkit-	5.0	10.5	3.2 -webkit-	4.0	11.0	2.1 -webkit-	2.2

Fig. 3.19 Tableau de compatibilité de border-radius

3.4.2. Dégradation gracieuse

Comme pour box-shadow, et afin d'offrir un support maximal de coins arrondis, border-radius peut être préfixée, comme suit :

```
div{
        -webkit-border-radius: 20px;
           -moz-border-radius: 20px;
               border-radius: 20px;
}
```

En revanche, et c'est encore plus vrai que pour box-shadow, vérifiez si l'ajout de ces préfixes est primordial puisque border-radius est compatible depuis assez longtemps en version non préfixée pour être utilisée tel quel. La majorité des navigateurs supportent la syntaxe officielle, les autres n'ajouteront simplement pas les coins arrondis.

Nous vous recommandons de ne plus utiliser les versions préfixées de border-radius, ni même d'autres techniques basées sur JavaScript. Cependant, des solutions existent. Certaines sont détaillées dans le chapitre 9 *Outils et ressources CSS*.

Note : plusieurs versions de navigateurs n'implémentent pas la même syntaxe concernant les règles longues de définition d'angle arrondi (par exemple, Firefox utilisait les valeurs border-radius suivies de -topleft, -topright, -bottomleft et -bottomright jusque dans sa version 3.6). C'est pour cela que nous vous déconseillons l'utilisation de ces propriétés longues pour définir chaque coin. Préférez l'utilisation raccourcie, tel que border-radius: 0 20px 0 0; pour définir un rayon de 20 px pour le coin en haut à droite.

3.5. Les images d'arrière-plan

L'utilisation des images d'arrière-plan en CSS est une fonctionnalité primordiale. Toutes les images concernant le design ne doivent pas être présentes sous forme de balises img dans le code HTML. Ces images n'apportent effectivement pas d'indications essentielles à une bonne compréhension du contenu. Elles n'ajoutent pas de signification à la page, elles se contentent de « rendre joli ». Pour appliquer nos images en CSS, nous disposons de plusieurs règles :

- background-image ou background pour les blocs ;
- list-style-image ou list-style pour les éléments de liste à puces ;
- border-image pour les images de bordures ;
- content pour le contenu généré (cas plus rare).

CSS3 apporte de nombreuses nouveautés concernant la gestion de ces images d'arrière-plan. Il est par exemple possible de choisir l'origine du positionnement d'une image appliquée en arrière-plan, de rogner le rendu de celle-ci, ou même de jouer sur sa taille. Mais la principale innovation est sans doute la gestion de plusieurs arrière-plans pour un même élément HTML.

3.5.1. Arrière-plans multiples

Pour utiliser plusieurs images d'arrière-plan sur un même élément, il suffit de les séparer par une virgule (,) dans la définition de la règle CSS (figure 3.20). Il n'est pas obligatoire de définir des fonds multiples composés uniquement d'images, la combinaison d'images et de couleurs est également possible, mais les arrière-plans multiples prennent tout leurs sens quand il s'agit d'appliquer plusieurs images en fond d'un élément unique.

```
background: url('image.png'), url('image2.png');
```

chemin vers l'image | chemin vers l'image 2

Fig. 3.20 Syntaxe pour l'utilisation des arrière-plans multiples

Avec la syntaxe des fonds multiples, les images sont appliquées de haut en bas, c'est-à-dire que la dernière image de la liste est masquée par les autres.

Les cas d'utilisations de cette fonctionnalité peuvent êtres multiples, mais l'une des raisons essentielle pourrait être l'allégement du poids des images. Une petite explication s'impose...

Fig. 3.21 Image à réaliser

Nous voulons réaliser l'image de la figure 3.21 pour une utilisation en arrière-plan d'une boîte. Nous remarquons que cette image est composée d'un fond texturé qui se répète, et d'une illustration. Avec la notation CSS3, il est tout à fait possible de créer deux images (figure 3.22) : une pour la texture et

une autre pour l'illustration.

Fig. 3.22 Deux images utilisées sur un arrière-plan multiple

Le code CSS est alors le suivant :

Ajout de deux images

```
div{
	width: 485px;
	height: 300px;
	background: url('illustration.png') no-repeat center, url('texture.png');
}
```

Si nous devions utiliser une image d'arrière-plan unique, le poids de celle-ci serait de 33 Ko (PNG optimisé) alors que là nos deux images pèsent 12 Ko + 1 Ko soit une économie de 20 Ko ! De plus, si l'élément devient plus grand ou plus petit, l'illustration reste toujours affichée au centre.

Arrière-plans multiples	9.0	3.6	4.0	3.1	10.5	3.2	10.0	2.1

Fig. 3.23 Tableau de compatibilité des arrière-plans multiples

3.5.2. background-repeat, background-position, background-attachment

Les trois propriétés background-repeat, background-position et background-attachment présentes dans la recommandation CSS2.1 sont donc modifiées pour êtres compatibles avec les arrière-plans multiples. Il faut alors séparer les valeurs par une virgule (,). L'ordre de définition doit être le même que l'ordre d'affectation des images. L'exemple précédent pourrait être écrit de cette façon :

```
div{
	width: 485px;
	height: 300px;
	background-image: url('illustration.png'), url('texture.png'),
	background-repeat: no-repeat, repeat;
	background-position: center, top left;
}
```

Pour background-repeat, deux nouvelles méthodes de répétition sont ajoutées, à l'aide des mots-clés space et round. Ces deux méthodes permettent de répéter l'image d'arrière-plan afin de remplir précisément le fond d'une boîte. La différence réside dans la méthode de remplissage :

- space ajoute des espaces vides entre les répétitions ;
- round redimensionne la taille des images.

Pour background-position, la spécification offre la possibilité d'écrire trois ou quatre valeurs, séparées par des espaces, là où CSS2.1 n'en proposait que deux. Ces nouvelles valeurs doivent être précédées d'un mot-clé qui définit le côté d'où s'applique le décalage. Ainsi, la déclaration background-position: bottom 10px right 20px; décale une image d'arrière-plan de 10 px du bas et de 20 px de la droite de la boîte. Pour info, le décalage par défaut s'effectue par rapport au coin en haut à gauche.

background-position (4 valeurs)	9.0	13.0			10.5		N.C	
background-repeat (round et space)	9.0				10.5		N.C	

Fig. 3.24 Tableau de compatibilité de background-position et background-repeat en CSS3. L'implémentation est en cours dans Webkit.

3.5.3. background-clip

La propriété background-clip permet de définir les limites auxquelles une image d'arrière-plan est rognée (clip en anglais) et donc n'est plus visible. La propriété à trois valeurs possibles :

- border-box : le fond est visible jusqu'aux limites extérieures de la bordure (mais ne la superpose pas) ;
- padding-box : le fond est visible jusqu'aux limites extérieures de la marge interne (padding), il s'arrête donc juste avant la bordure ;
- content-box : le fond est visible uniquement sous le contenu. Il s'arrête à la limite intérieure de la marge interne. Cela correspond à la taille spécifiée par width et height dans le modèle de boîte classique.

La valeur par défaut est background-clip: border-box.

Visualisons cet effet avec une bordure et des marges internes sur une div (figure 3.25)

```
div{
        width: 485px;
        height: 300px;
        border: 20px solid rgba(255, 255, 255, 0.3);
        padding: 40px;
        background-image: url('illustration.png'), url('texture.png');
}
```

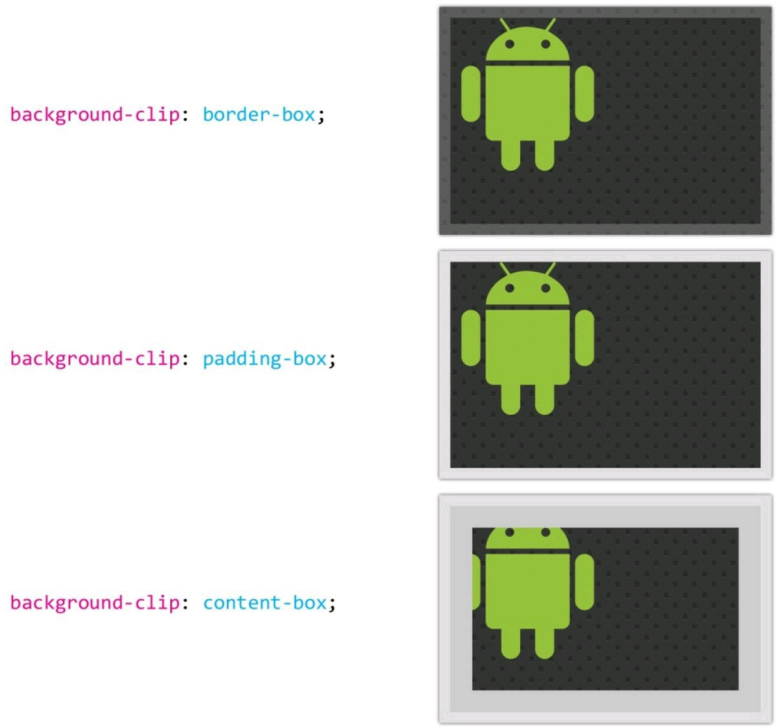


Fig. 3.25 Réglages de background-clip

background-clip	9.0	3.6 -moz-	4.0	4.0	5.0	10.0 -o-	10.5	4.0	10.0	2.2

Fig. 3.26 Tableau de compatibilité de background-clip.
Dans Firefox 3.6, les valeurs sont border et padding. content n'existe pas.

Note : le moteur Webkit (uniquement) inclut une quatrième valeur pour background-clip, la valeur text. Celle-ci permet de visualiser un arrière-plan au travers du texte uniquement, seulement si la couleur du texte est transparente. Pour éviter des problèmes d'accessibilité, la propriété -webkit-text-fill-color permet de surpasser la valeur de la propriété color. La valeur text et la propriété -webkit-text-fill-color ne sont pas définies au sein du W3C, même si le rendu visuel est très intéressant.

3.5.4. background-origin

La propriété background-origin permet de définir l'origine du positionnement initial des images d'arrière-plan. La propriété à trois valeurs possibles, qui sont similaires à celles de background-clip :

- border-box : l'origine est la limite externe de la bordure ;
- padding-box : l'origine est la limite externe de la marge interne ;
- content-box : l'origine est la limite externe du contenu.

La valeur par défaut est background-origin: padding-box (figure 3.27).

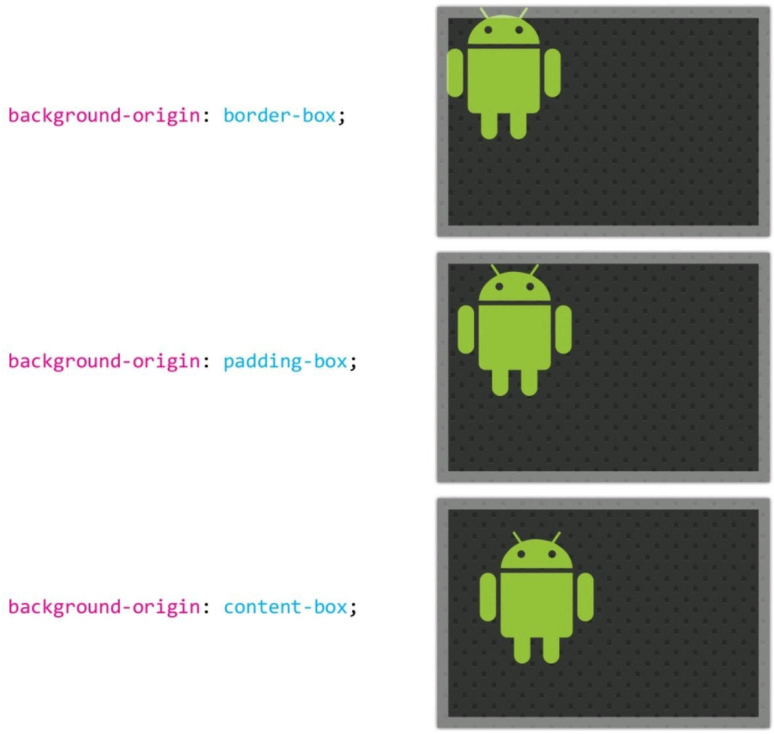


Fig. 3.27 Réglages de background-origin

background-origin	9.0	3.6 -moz-	4.0	4.0	3.1 -webkit-	5.0	10.0 -o-	10.5	3.2 -webkit-	4.0	10.0	2.1 -webkit-	2.2

Fig. 3.28 Tableau de compatibilité de background-origin.
Dans Firefox 3.6, les valeurs sont border, padding et content.

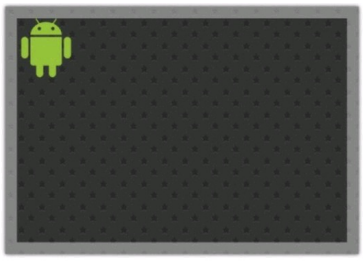
3.5.5. background-size

La propriété background-size permet de spécifier la taille de l'image appliquée en arrière-plan. Les valeurs possibles sont :

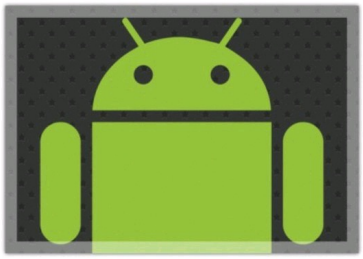
- valeurs numériques pour définir la largeur et/ou la hauteur. Le ratio est conservé si une seule valeur est fournie ;
- pourcentage : par rapport à la boîte elle-même. Une valeur de 100 % couvre la largeur de la boîte et le ratio est conservé. Une valeur de 100 % 100 % couvre toute la boîte, mais le ratio de l'image n'est pas conservé ;
- contain : le fond est étendu à la taille de la boîte en ajustant le plus grand côté (largeur ou hauteur) au sein de celle-ci. Le ratio de l'image est conservé ;
- cover : le fond est étendu à la taille de la boîte en ajustant le plus petit côté (largeur ou hauteur) au sein de celle-ci. Le ratio de l'image est conservé.

La valeur par défaut est background-size: auto. Toutes les valeurs omises correspondent à auto.

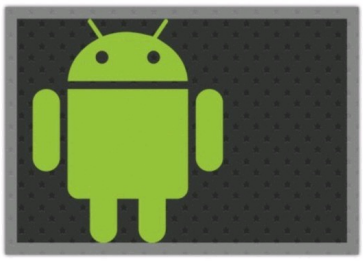
```
background-size: 100px;
```



```
background-size: cover;
```



```
background-size: contain;
```



```
background-size: 100% 100%;
```

Fig. 3.29 Réglages de background-size

Attention, une pixellisation de l'image d'arrière-plan est visible si l'agrandissement est trop important. C'est le cas notamment lorsque nous demandons à une image d'occuper tout l'arrière-plan (avec background-size:100% 100% par exemple) alors que celle-ci est plus petite (figure 3.29).

background-size	9.0	3.6 -moz-	4.0	4.0	3.1 -webkit-	5.0	10.0 -o-	10.5	3.2 -webkit-	4.0	10.0	2.1 -webkit-	2.2

Fig. 3.30 Tableau de compatibilité de background-size

3.5.6. SVG

Sur le web, les trois formats d'images actuellement disponibles sont GIF, JPG et PNG. Ces formats ont un point commun, ce sont des formats bitmaps, c'est-à-dire qu'ils sont composés de pixels (plus précisément d'une matrice de pixels). Ils peuvent être utilisés depuis la balise img, ou depuis le CSS *via* background-image par exemple (voir compatibilité figure 3.30).

Depuis peu, un quatrième format s'invite à la fête. C'est le SVG. Ce format est, contrairement aux autres, un format vectoriel, ce qui implique qu'il n'est pas composé de pixels, mais il est construit à partir de formes géométriques, lesquelles sont définies par des attributs de tailles, de positions, de couleurs...

La principale différence entre ces deux types d'images est qu'une image vectorielle est

redimensionnable sans perte de qualité, ce qui n'est pas le cas d'une image bitmap (figure 3.31). De plus, une image vectorielle est souvent plus légère (en termes de poids).

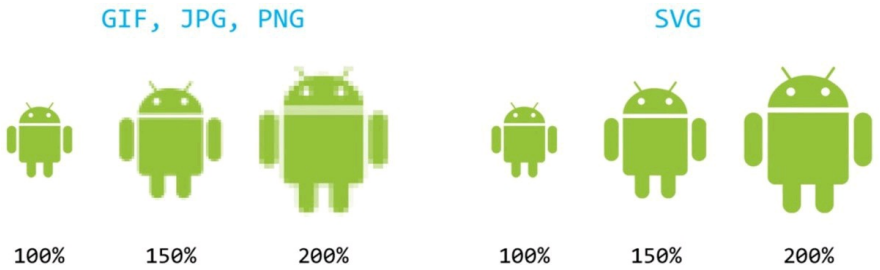


Fig. 3.31 Résultat visuel de l'agrandissement d'une image bitmap vs vectorielle

Il devient donc possible d'utiliser une image SVG dans une balise HTML img, ou depuis une propriété CSS. Les propriétés d'arrière-plans permettent ensuite de modifier cette image, à la manière d'une image classique. Il n'est donc plus obligatoire de faire attention aux agrandissements trop importants, puisqu'aucune pixellisation ne sera provoquée.

```
div{
	background-image: url('image.svg');
	background-size: 100%;
}
```

Dans cet exemple, nous appliquons une image d'arrière-plan en SVG, en modifiant sa taille pour qu'elle s'adapte au bloc, sans se soucier de sa taille réelle. Le résultat sera parfait dans tous les cas.

SVG en CSS	9.0	4.0	5.0	3.2	9.5	3.2	10.0	3.0
SVG dans <img>	9.0	4.0	4.0	3.2	9.0	3.2	10.0	3.0

Fig. 3.32 Tableau de compatibilité de l'intégration de SVG

3.6. Les bordures en images

La propriété border-image permet d'ajouter des images sur les bordures d'une boîte. Cette propriété est la notation raccourcie des propriétés border-image-source, border-image-slice, border-image-width, border-image-outset et border-image-repeat. L'utilisation de border-image ne modifie pas la taille de l'élément, la bordure peut alors apparaître à l'extérieur de la boîte.

Le principe d'application est assez simple : une image unique doit être utilisée pour définir les différents morceaux de la bordure. L'image doit être composée de neuf parties, qui définissent les quatre coins, les quatre parties étirables et le centre de la boîte (figure 3.33).

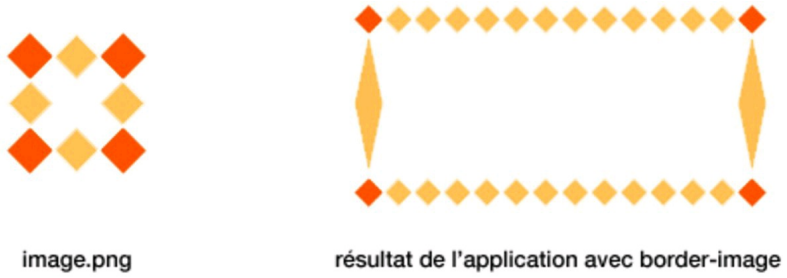


Fig. 3.33 Exemple sur le site du W3C

Voyons à présent en détail l'utilisation des différentes règles, mais attention, l'implémentation de border-image étant encore incomplète, voire approximative, l'état actuel décrit dans ce livre est celui présent dans la spécification du W3C. Il se peut que les navigateurs ne comprennent que peu, ou pas, les règles présentées. Vous êtes prévenus...

3.6.1. border-image-source

La propriété border-image-source permet de définir l'image utilisée pour la création de la bordure. Il faut en spécifier la source, d'une façon classique, comme pour background-image.

```
div{
	border-image-source: url('dossier/image.png');
}
```

3.6.2. border-image-slice

La propriété border-image-slice permet de définir les traits de découpe des parties de l'image (figure 3.34). Cette règle accepte une, deux, trois ou quatre valeurs séparées par des espaces qui permettent de découper l'image depuis le haut, depuis la droite, depuis le bas, et depuis la gauche, en utilisant le principe de définition des marges.

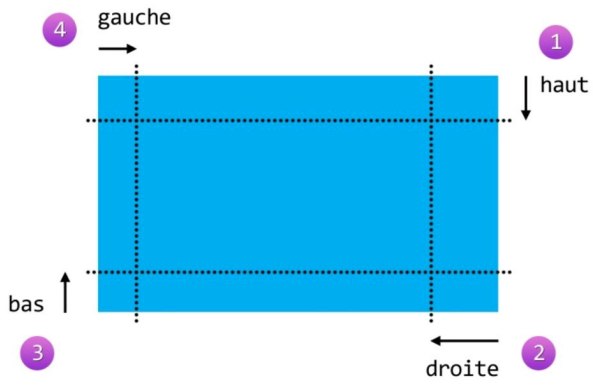


Fig. 3.34 Principe de découpage avec border-image-slice

Ainsi, sur ce bloc mesurant 130 px × 95 px, les traits de découpe seront exprimés comme suit :

```
div{
	border-image-slice: 18px 24px 16px 45px;
}
```

La présence du mot-clé fill permet de spécifier la partie centrale de l'image comme étant l'arrière-plan de la boîte. Il est donc possible de gérer les bordures et le fond d'une boîte en une seule image.

3.6.3. border-image-width

La propriété border-image-width permet de définir la largeur ou la proportion de la bordure. Les valeurs possibles sont :

- une, deux, trois ou quatre valeurs qui définissent la largeur de chaque bordure dans l'ordre classique ;
- nombre : correspond à la proportion de la taille de la bordure actuelle, définie avec border-width ;
- auto : la largeur des bordures est calculée pour s'adapter à la taille des parties de l'image définies avec border-image-slice.

La valeur par défaut est border-image-width: 1. Il faut donc que border-width soit définie pour que la bordure s'affiche.

3.6.4. border-image-outset

La propriété border-image-outset permet de définir le décalage des éléments constituants la bordure, par rapport à la limite de bordure classique. Les valeurs possibles sont :

- une, deux, trois ou quatre valeurs qui définissent le décalage de chaque bordure dans l'ordre classique ;
- nombre : le décalage correspond à la proportion de la taille de la bordure actuelle, définie avec border-width.

La valeur par défaut est border-image-outset: 0.

3.6.5. border-image-repeat

La propriété border-image-repeat permet de définir la méthode de répétition des éléments de l'image. Les coins ne sont pas affectés. Les valeurs possibles sont :

- stretch : l'image est étirée pour remplir la bordure.
- Repeat : l'image est répétée pour remplir la bordure.
- round : idem à repeat. En revanche, l'image peut être redimensionnée pour obtenir un nombre précis de répétition.
- space : idem à repeat. En revanche, les images peuvent être espacées pour obtenir un nombre précis de répétition. Il ne se produit aucun redimensionnement.

Deux mots-clés peuvent êtres utilisés pour spécifier un comportement différent entre les côtés verticaux et horizontaux de la bordure.

La valeur par défaut est border-image-repeat: stretch.

3.6.6. border-image: notation raccourcie

La notation raccourcie permet de spécifier en une seule ligne les différents paramètres de création de la bordure :

```
div{
border-image: url('image.png') 27px / 27px / 0 repeat stretch;
}
```

où chaque règle est définie ainsi :

```
border-image: source slice / width / outset repeat;
```

3.6.7. Dans les faits...

Comme nous l'avons déjà dit, le support de border-image et de ses dérivés est insuffisant. Voici notamment ce qu'il faut retenir pour l'ensemble des navigateurs actuels :

- seule la notation raccourcie est interprétée, et de plus elle doit être préfixée ;
- border-image-slice doit être défini sans unités ;
- border-image-width n'est pas reconnue dans Opéra ;
- border-image-outset n'est reconnue par aucun navigateur ;
- le mot-clé fill sur border-image-slice est inconnu, et fait même planter certains navigateurs ;
- le mot-clé space sur border-image-repeat est inconnu ;
- IE10 ne reconnaît toujours pas border-image.

Voici cependant quelques exemples utilisables dans les navigateurs du marché. L'image utilisée est la suivante (figure 3.35), une image de 300 px sur 394 px.

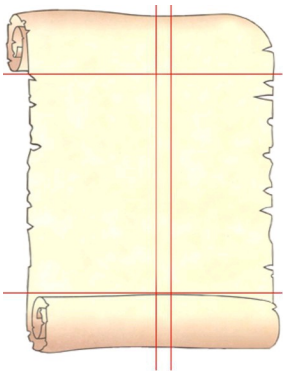


Fig. 3.35 Image utilisée avec border-image (les traits de découpes sont en rouge)

Application de l'image avec une taille de bordure de 100 px, les découpes sont :

- Haut : 74 px
- Droite : 120 px
- Bas : 84 px
- Gauche : 164 px

Ajout de l'image en bordure

```
div{
	border-width: 100px;
	border-image: url('image.png') 74 120 84 164;
}
```

est équivalent à :

```
div{
	border-image: url('image.png') 74 120 84 164 / 100px;
}
```

Modifions alors la taille de chaque bordure indépendamment :

```
div{
	border-image: url('image.png') 74 120 84 164 / 60px 70px 60px 100px;
}
```

Puis testons les différents modes de répétition (figure 3.36) :

Répétition stretch

```
div{
	border-image: url('image.png') 74 120 84 164 / 60px 70px 60px 100px stretch;
}
```

Répétition round

```
div{
	border-image: url('image.png') 74 120 84 164 / 60px 70px 60px 100px round;
}
```

Dernier exemple, les largeurs des bordures sont deux fois plus petites, mais les découpes restent identiques (figure 3.36) :

```
div{
	border-image: url('image.png') 74 120 84 164 / 30px 35px 30px 50px round;
}
```

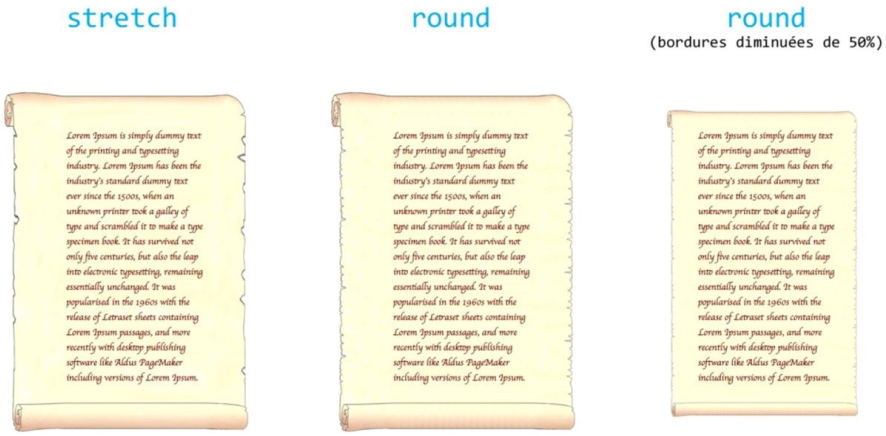


Fig. 3.36 Différents exemples d'applications de border-image

La dégradation gracieuse est fortement encouragée pour l'utilisation de border-image, car son support complet reste encore très anecdotique.

border-image		3.5 -moz-	4.0 -webkit-	16.0	3.1 -webkit-	11.0 -o-	3.2 -webkit-	11.0 -o-	2.1 -webkit-

Fig. 3.37 Tableau de compatibilité de border-image

3.7. Les dégradés

L'utilisation des dégradés sur le web est, comme pour les ombres, une manière de donner du relief à un graphisme. Les dégradés, à condition qu'ils soient bien utilisés, sont quasiment non visibles pour l'utilisateur, mais son utilisation est nécessaire pour que les éléments d'une page ne paraissent plus

plats (figure 3.38). Ils sont très souvent employés pour la création de boutons, d'éléments de menu ou encore en arrière-plan d'un site, et souvent utilisés conjointement avec des ombres pour accentuer encore l'effet de lumière.

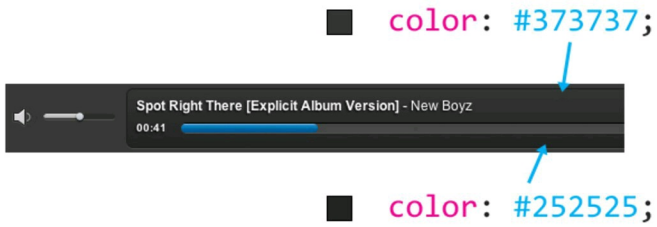


Fig. 3.38 Un dégradé CSS sur le player du site de deezer.com

Les dégradés CSS ont été historiquement soumis au W3C par Apple en 2008, qui utilisait alors une syntaxe très différente de celle qui est actuellement dans la spécification. L'ancienne construction peut encore être présente sur certains sites, ou expliquée dans d'anciens articles. Nous vous recommandons de ne pas, ou plus utiliser cette écriture, mais de préférer la dégradation gracieuse de votre design. En effet, cette syntaxe est vouée à disparaître et il n'est pas nécessaire de continuer à la maintenir.

La création des dégradés décrite dans ce livre suit l'état de la spécification telle qu'elle était en septembre 2011. Depuis cette date, des changements ont eu lieu, mais par manque de support au sein des navigateurs, nous avons choisi de vous présenter « l'ancienne » syntaxe. Reportez-vous en fin de chapitre pour une explication des différences.

Après cette petite explication, commençons par comprendre ce que sont les dégradés. Dans la pratique, les dégradés CSS sont des images créées « à la volée » qui représentent une transition douce entre différentes couleurs. Pour l'instant, il n'en existe que deux types, les dégradés linéaires et les dégradés radiaux. Le W3C définit deux fonctions, linear-gradient() et radial-gradient(), qui nous permettent de créer ces dégradés, et comme le résultat est une image, ces fonctions sont utilisables avec les mêmes propriétés que celles qui acceptent des images, à savoir:

- background-image ou background
- list-style-image ou list-style
- border-image
- content

Il existe également deux fonctions supplémentaires, repeating-linear-gradient() et repeating-radial-gradient() qui sont, comme leurs noms l'indiquent, des dégradés linéaires ou radiaux qui se répètent.

3.7.1. linear-gradient

La fonction linear-gradient() permet de créer un dégradé linéaire et prend trois paramètres où les valeurs sont dans l'ordre (figure 3.39) :

- **direction de la ligne du dégradé** (optionnel) qui peut être :
  - un mot-clé (top, bottom, left ou right) qui indique le côté d'où commence le dégradé, en direction du côté opposé.
  - deux mots-clés (ex. : top left) qui indiquent l'angle d'où commence le dégradé, en direction de l'angle opposé (ici bottom right).

- un angle, inclinaison de la ligne qui passe par le centre de l'élément. Par exemple, une valeur de 0deg est identique à left, une valeur de 90deg est identique à bottom.

- **couleur de départ** (obligatoire) ;
- **couleur de fin** (obligatoire).

Fig. 3.39 Syntaxe des dégradés linéaires

La direction par défaut d'un dégradé est top, ce qui signifie que le dégradé est créé depuis le haut vers le bas si aucune direction n'est spécifiée, comme dans l'exemple suivant (figure 3.40) :

```
#element{
	background: linear-gradient( deepskyblue, greenyellow);
}
```

Fig. 3.40 Rendu de linear-gradient

Nous pouvons noter que les dégradés linear-gradient(deepskyblue, greenyellow) et linear-gradient(bottom, greenyellow, deepskyblue) sont identiques !

Encore un exemple, pour un dégradé linéaire allant de l'angle en haut à gauche, vers l'angle en bas à droite (figure 3.41) :

```
#element{
	background: linear-gradient( top left, tomato, plum);
}
```

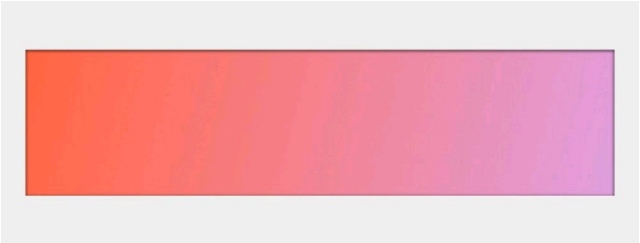


Fig. 3.41 Rendu de linear-gradient

Et voici un dégradé linéaire utilisant un angle de 45 degrés (figure 3.42) :

```
#element{
	background: linear-gradient( 45deg,  green, lightgreen);
}
```

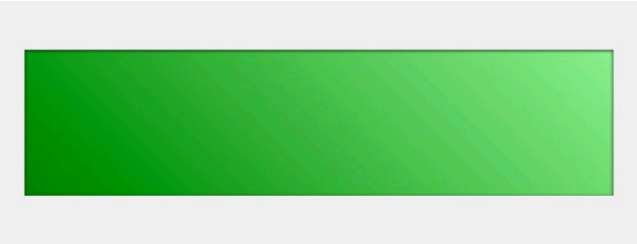


Fig. 3.42 Rendu de linear-gradient

Attention, l'utilisation d'un dégradé avec un angle de 45 degrés n'est pas identique à la valeur bottom left, sauf dans le cas où l'élément sur lequel est appliqué le dégradé est carré.

Concernant l'utilisation des couleurs, les choix classiques sont possibles. Il est donc possible d'utiliser :

- l'un des mots-clés du langage : red, green, darkkhaki...
- un code hexadécimal : #38FF5F, #CCC...
- une fonction de couleur avec ou sans transparence : rgb(255,0,28), rgba(0,0,0,0.5)...
- le mot-clé transparent : équivalent à rgba(0,0,0,0).

Il est également possible d'utiliser plusieurs dégradés sur un même élément, *via* les arrière-plans multiples, ou même d'utiliser un dégradé et une image. Bien entendu, le dégradé de premier plan doit être transparent, au moins en partie, pour laisser apparaître le deuxième plan. Le tutoriel en fin de chapitre utilise ces techniques.

Question compatibilité, nous pouvons noter que la majorité des navigateurs récents supportent les dégradés linéaires, mais en version préfixée. Le code suivant est alors nécessaire pour un support complet.

Syntaxe complète pour un dégradé

```
#element{
    /* anciens navigateurs */
    background-color: green;
    background-image: url('degrade.png');
    /* chrome, safari */
    background: -webkit-linear-gradient( lightgreen, green);
    /* firefox */
    background:    -moz-linear-gradient( lightgreen, green);
    /* internet explorer */
    background:     -ms-linear-gradient( lightgreen, green);
    /* opéra */
    background:      -o-linear-gradient( lightgreen, green);
    /* standard */
    background:         linear-gradient( lightgreen, green);
}
```

L'ajout de règles background avant le dégradé permet de prévoir une dégradation gracieuse sur les anciens navigateurs (figure 3.43). Cette règle sera écrasée uniquement par les navigateurs qui comprennent au moins l'une des syntaxes qui suivent (avec dégradés). Dans notre cas, nous précisons une couleur et/ou une image en solution de repli. L'image n'est pas obligatoire dans le cas d'un dégradé subtil. De plus, notez que la propriété background utilisée pour les dégradés écrase background-color et background-image.

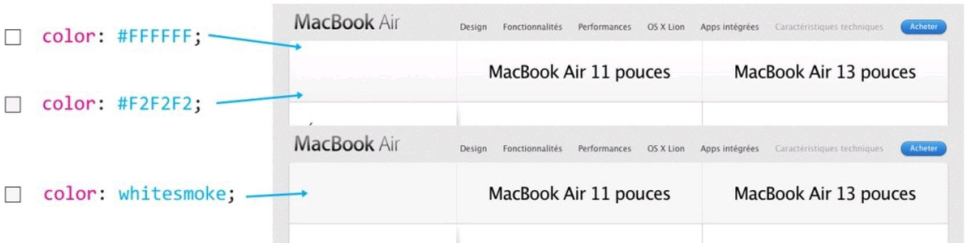


Fig. 3.43 Dégradation gracieuse de l'utilisation des dégradés.
Les dégradés linéaires peuvent êtres simulés avec des filtres propriétaires pour les anciennes versions d'IE.

Dégradés linéaires	10.0 -ms-	3.6 -moz-	4.0 -webkit-	16.0	4.0 -webkit-	11.1 -o-	3.2 -webkit-	11.1 -o-	2.1 -webkit-

Fig. 3.44 Tableau de compatibilité des dégradés linéaires

3.7.2. radial-gradient

La fonction radial-gradient() permet de créer un dégradé radial, et prend quatre paramètres où les valeurs sont dans l'ordre (figure 3.45) :

- **position du centre** (optionnel) à la manière de background-position :
  - un mot-clé (top, bottom, left, right ou center) ;
  - deux mots-clés (ex. : top left) ;
  - valeur numérique (ex. : 50 px, 100 px, 50 %, 50 %).
- définition de la **taille** et de la **forme** de l'ellipse (optionnel) :
  - choix de la forme: circle ou ellipse,
  - choix de la taille :
  - closest-side, closest-corner, farthest-side, farthest-corner, contain, cover ;
  - valeurs numériques par rapport au centre déjà défini.
- **couleur de départ** (obligatoire) ;
- **couleur de fin** (obligatoire).

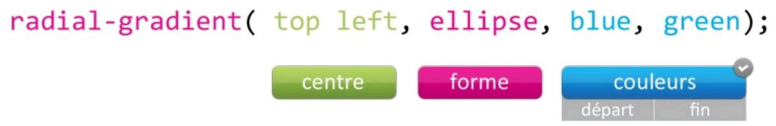


Fig. 3.45 Syntaxe des dégradés radiaux

Par défaut, si le centre n'est pas précisé et si la taille et/ou la forme ne l'est pas non plus, le dégradé est alors une ellipse placée au centre de l'élément. Voici la syntaxe minimale pour la création d'un dégradé radial (figure 3.46) :

```
#element{
        background: radial-gradient( yellow, deeppink);
}
```

Fig. 3.46 Rendu de radial-gradient

Voici un exemple de dégradé où le centre a été positionné en haut à gauche de l'élément (figure 3.47) :

```
#element{
	background: radial-gradient(top left, aquamarine, dodgerblue);
}
```

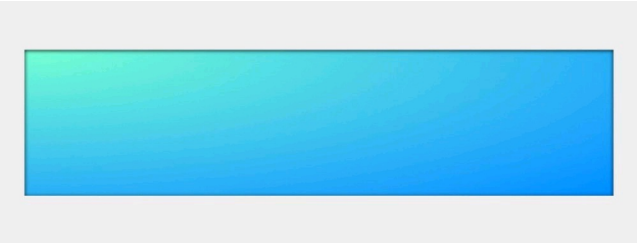


Fig. 3.47 Rendu de radial-gradient

Par défaut, le dégradé est en forme d'ellipse et sa taille est de type farthest-side. C'est-à-dire que l'ellipse est dessinée jusqu'au côté qui est le plus éloigné du centre. Le rendu visuel est alors un quart de cette ellipse.

Parcourons les différentes options qui s'offrent à nous. Tout d'abord, pour définir la forme du dégradé, nous disposons de deux mots-clés: circle et ellipse. Rien de bien méchant, soit c'est un rond, soit une ellipse. Pour définir la taille, nous avons plusieurs choix. Première possibilité, utiliser des valeurs numériques (figure 3.48)

```
#element{
	background: radial-gradient(top left, 100px 100px, aquamarine, dodgerblue);
}
```

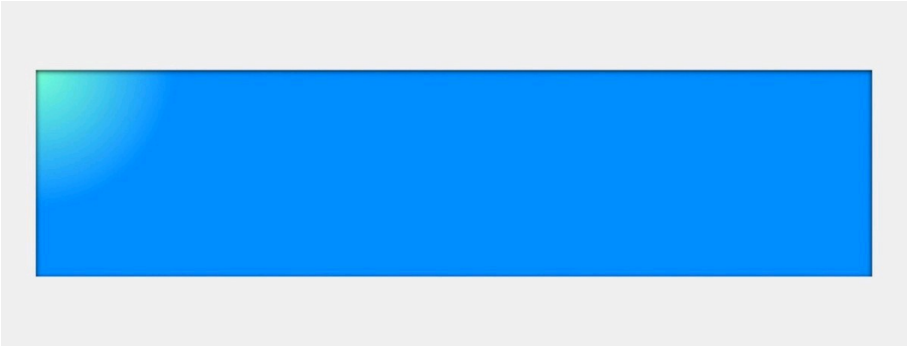


Fig. 3.48 Rendu de radial-gradient

Dans ce cas, un cercle de 100 px de rayon est créé, et celui-ci est positionné dans l'angle en haut à gauche. Attention, seul Chrome reconnaît cette écriture. Les autres navigateurs ne créent pas de

dégradé.

Il nous reste alors les mots-clés suivants qui nous permettent de définir l'extrémité de l'ellipse (figure 3.49, figure 3.50) :

- closest-side: vers le côté le plus proche ;
- closest-corner: vers le coin le plus proche ;
- farthest-side: vers le côté le plus éloigné ;
- farthest-corner: vers le coin le plus éloigné ;
- contain: synonyme de closest-side. L'ellipse est entièrement visible, contenue ;
- cover: synonyme de farthest-corner. L'ellipse couvre généralement l'ensemble.

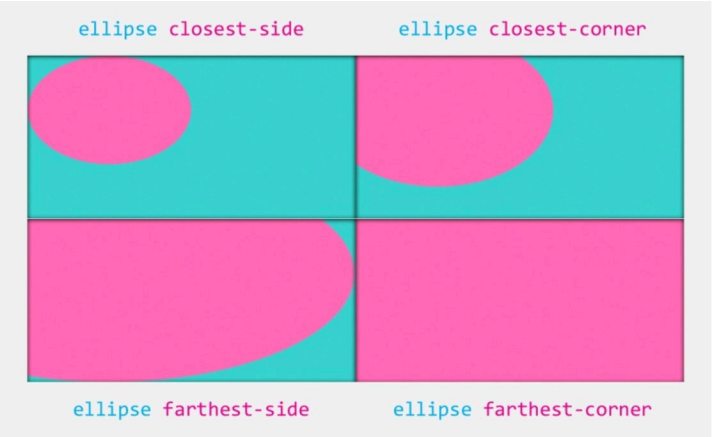


Fig. 3.49 Différents dégradés elliptiques

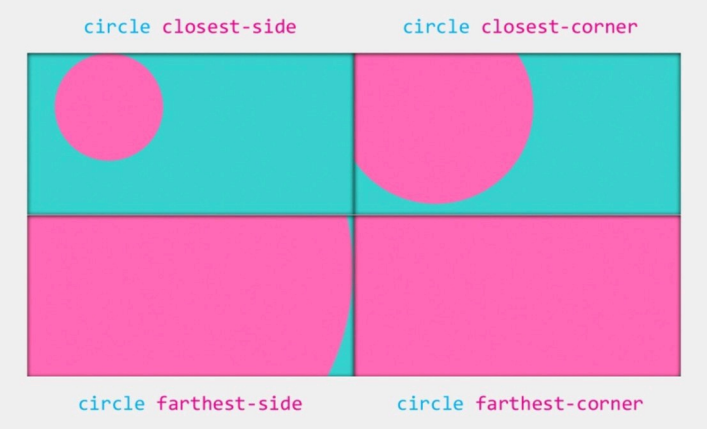


Fig. 3.50 Différents dégradés ronds

Note : pour plus de clarté et de compréhension, nous avons représenté la transition de couleur par une limite précise. Nous verrons la syntaxe plus tard.

Pour terminer, nous pouvons dire que les dégradés radiaux restent encore assez mal supportés au sein des navigateurs. Par exemple, Chrome ne reconnaît le mot-clé ellipse que depuis peu de temps, ce qui signifie que seuls des cercles peuvent êtres créés. Mais par contre il est le seul navigateur à supporter les valeurs numériques pour la fin de l'ellipse. De plus, comme pour les dégradés linéaires, les versions préfixées doivent être ajoutées pour un support dans chaque navigateur. Vous l'aurez compris, le plus efficace est encore d'utiliser les dégradés radiaux avec la syntaxe la plus simple possible, mais surtout de penser à la dégradation gracieuse.

Dégradés radiaux	10.0 -ms-	3.6 -moz-	4.0 -webkit-	16.0	4.0 -webkit-	11.6 -o-	3.2 -webkit-	12.0 -o-	2.1 -webkit-

Fig. 3.51 Tableau de compatibilité des dégradés radiaux.
Les dégradés radiaux peuvent êtres simulés avec des filtres propriétaires pour les anciennes versions d'IE.

3.7.3. color-stop

Les *color-stops* (littéralement les couleurs d'arrêt) sont des points placés sur la ligne d'un dégradé qui nous permettent d'ajouter de nouvelles couleurs, et ainsi créer un dégradé composé de plus de deux couleurs. Ils sont utilisables aussi bien avec des dégradés linéaires que radiaux. Les *color-stops* sont obligatoirement définis dans l'ordre, et une position peut être précisée en même temps que la définition de la couleur. Ainsi, ces deux dégradés sont différents (figure 3.52) :

```
#element{
	background: linear-gradient( darkviolet, deeppink, darkviolet);
}
#element2{
	background: linear-gradient( darkviolet, deeppink 80%, darkviolet);
}
```

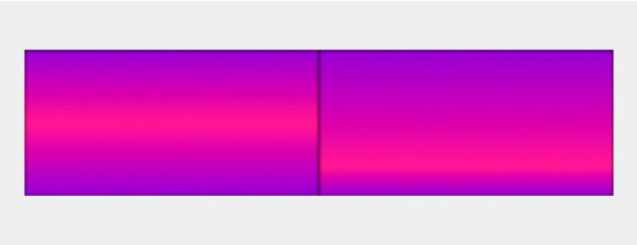


Fig. 3.52 Différence entre deux dégradés

Dans le premier exemple, comme aucune position n'est précisée pour les *color-stops*, les navigateurs se chargent alors de les calculer. Dans notre cas, la valeur deeppink est calculée à 50 %, puisque par défaut la première valeur vaut 0 % et la dernière 100 %. Les transitions s'effectuent donc de darkviolet 0 % à deeppink 50 %, puis de deeppink 50 % à darkviolet 100 %. Dans le second exemple, la couleur deeppink est fixée à 80 %, donc les transitions s'effectuent de 0 % à 80 %, puis de 80 % à 100 %.

Il est bien entendu possible de spécifier des *color-stops* de départ et de fin différents des valeurs par défaut (0 et 100 %). Dans ce cas, la couleur utilisée avant le premier *color-stop* est celle du premier *color-stop* et celle utilisée après le dernier *color-stop* est celle du dernier. La transition de couleur s'effectue classiquement entre les deux valeurs. Voici un exemple où le dégradé est effectif entre 30 et 50 % (figure 3.53) :

```
#element{
	background: linear-gradient( bottom left, cornflowerblue 30%, darkorchid 50%);
}
```

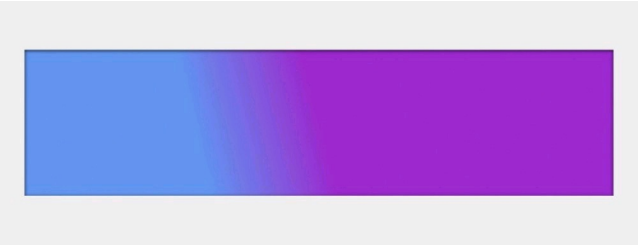


Fig. 3.53 Rendu d'un dégradé linéaire avec color-stop

Chose pratique, les *color-stops* peuvent êtres définis en dehors de la boîte. Ils peuvent donc être supérieurs à 100 % ou inférieurs à 0 %. Voici une comparaison de deux dégradés avec un *color-stop* à 150 % sur le deuxième (figure 3.54).

```
#element{
	background: linear-gradient( limegreen, yellow);
}
#element2{
	background: linear-gradient( limegreen, yellow 150%);
}
```

Fig. 3.54 L'étirement est visible

Pour terminer, il est également possible de définir deux *color-stops* consécutifs avec la même valeur. Dans ce cas, aucune transition douce de couleur n'est créée. Au contraire, le résultat est une limite franche, comme sur l'exemple suivant (figure 3.55) :

```
#element{
	background: linear-gradient( skyblue 30%, limegreen 30%, limegreen 70%, yellow 70%);
}
```

Fig. 3.55 Une limite franche est créée

3.7.4. repeating-linear-gradient, repeating-radial-gradient

Les fonctions repeating-linear-gradient() et repeating-radial-gradient() permettent de créer des dégradés linéaires ou radiaux qui se répètent, afin de créer une sorte de motif.

L'utilisation de ces fonctions est strictement identique aux fonctions raccourcies présentées précédemment, si ce n'est que l'utilisation d'au moins un *color-stop* (compris entre 0 et 100 %) est obligatoire. Voici un exemple de ce que nous pouvons faire avec les dégradés linéaires (figure 3.56) :

```
#element{
	background: repeating-linear-gradient( left, #CCC, #EEE 10%);
}
```

Fig. 3.56 Dégradé linéaire qui se répète

Ou encore avec les dégradés radiaux (figure 3.57) :

```
#element{
	background: repeating-radial-gradient( #CCC, #EEE 10%);
}
```

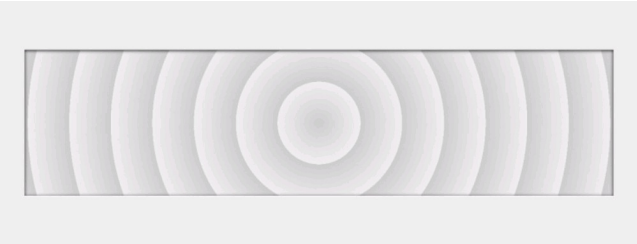


Fig. 3.57 Dégradé radial qui se répète

Il est possible de reproduire certains dégradés qui se répètent avec la syntaxe des dégradés classiques et en modifiant la taille de l'arrière-plan. Le premier exemple peut donc être reproduit ainsi:

```
#element{
	background: linear-gradient( left, #CCC, #EEE);
	background-size: 10%;
}
```

Cela fonctionne uniquement pour les dégradés linéaires, et s'ils sont définis sans angle.

3.7.5. Autres syntaxes

Ancienne syntaxe Webkit

Bien que nous vous déconseillons son utilisation, il peut être utile de comprendre son fonctionnement, et surtout, si vous y êtes confrontés, vous ne serez pas perdu. La syntaxe était donc la suivante :

```
#element{
	background: -webkit-gradient( linear, left top, left bottom, from(red), color-stop(green, 0.5), to(blue));
```

```
}
```

Cette écriture est l'équivalent de :

```
#element{
	background: -webkit-linear-gradient( top, red, green 50%, blue);
}
```

Vous comprenez aisément qu'une telle syntaxe n'est pas maintenable dans le temps.

Nouvelle syntaxe (septembre 2011)

Bien que la syntaxe définie par le W3C n'ait pas évolué pendant de long mois et que les navigateurs se soient emparés des dégradés, la spécification a été modifiée en septembre 2011. Et c'est bien le principe des brouillons de travail. Cette nouvelle syntaxe propose de clarifier la création des dégradés linéaires et radiaux.

Pour les dégradés linéaires, le mot-clé to fait son apparition. Ce mot-clé s'utilise conjointement avec les mots-clés de direction classique (top, left, right, bottom) et spécifie la destination du dégradé. En effet, nous créons un dégradé vers une direction, plutôt que depuis une direction, comme ceci :

Dégradé vers la droite

```
#element{
	background: linear-gradient( to right, skyblue, steelblue);
}
```

Cette notation est la nouvelle syntaxe de :

Dégradé depuis la gauche

```
#element{
	background: linear-gradient( left, skyblue, steelblue);
}
```

Pour les dégradés radiaux, c'est le mot-clé at qui est ajouté. Le sens d'écriture des éléments est inversé. Ce mot-clé permet donc de définir d'abord la taille et la forme, puis le centre, comme ceci :

```
#element{
	background: radial-gradient( farthest-side at top left, red, green);
}
```

Cette notation est la nouvelle syntaxe de :

```
#element{
	background: radial-gradient( top left, farthest-side, red, green);
}
```

À l'écriture de ces lignes, seuls Opéra 11.6 et Firefox 10 supportent la nouvelle notation pour les dégradés linéaires et aucun navigateur n'implémente celle des dégradés radiaux. Il est probable que le support se soit étendu depuis pour correspondre à la spécification.

3.8. À vous de jouer !

C'est à présent à votre tour. Nous allons mettre en pratique ce que nous venons de détailler dans ce chapitre, en créant la partie #intro du site de démonstration. Plus précisément, nous allons reproduire l'élément #graphikBloc, qui constitue l'élément graphique le plus important de la page. Pour vous mettre l'eau à la bouche, jetez un œil au site (figure 3.58).

Fig. 3.58 Partie #intro du site support

Commençons par créer le code HTML nécessaire :

Code HTML

```
<div id="graphikBloc"></div>
```

Voilà ! Le moins que l'on puisse dire, c'est que c'est condensé. Mais pourtant, tout est là, nous n'avons pas besoin de plus de balises, tout s'effectue donc en CSS. N'oubliez pas qu'aucun code d'exemple n'est préfixé.

Tout d'abord, définissons la taille de l'élément, et ajoutons-lui des bordures :

CSS

```
#graphikBloc{
	width: 500px;
	height: 310px;
	border: 10px solid #181818;
}
```

Complétons avec les coins arrondis (figure 3.59) :

```
#graphikBloc{
	...
	border-radius: 3px;
}
```

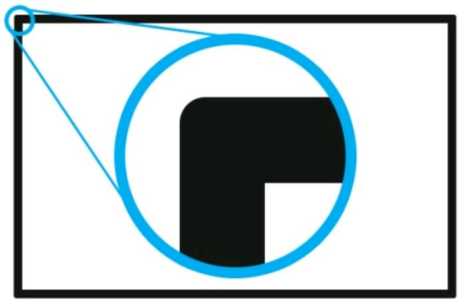


Fig. 3.59 Notre bloc a des coins arrondis

Créons maintenant l'arrière-plan. Celui-ci est composé de deux dégradés subtils qui simulent la profondeur, du personnage puis d'une image de texture (figure 3.60).

```
#graphikBloc{
        ...
background: linear-gradient(left,black,transparent 1.5%,transparent 98.5%, black),
            linear-gradient(black,transparent 1.5%, transparent 98.5%, black),
            url('bugdroid.svg'),
            url('etoiles.png');
}
```

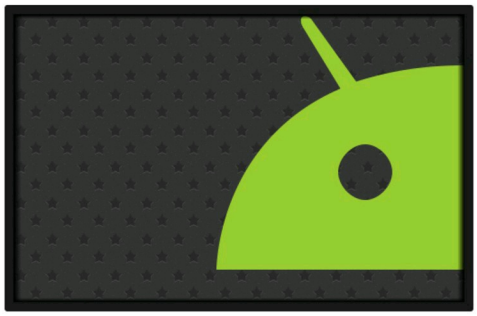


Fig. 3.60 Notre bloc à quatre arrière-plans

Le premier fond est un dégradé linéaire qui est créé de gauche à droite, avec une valeur de noir en position 0 % jusqu'à transparent à 1,5 %, puis de transparent à 98,5 % jusqu'à noir en position 100 %. Le deuxième utilise le même principe mais le dégradé est créé du haut vers le bas.

Pour le troisième arrière-plan, qui est une image SVG, le personnage est beaucoup trop grand. Nous allons donc le redimensionner avec background-size. Nous réduisons également la taille des étoiles. Mais attention, nous modifions uniquement la taille du dernier et de l'avant dernier arrière-plan (figure 3.61)

```
#graphikBloc{
        ...
        background-size: auto, auto, 160%, 5%;
}
```

Fig. 3.61 Le personnage et les étoiles sont redimensionnés

Pour rappel, la valeur de background-size en pourcentage se réfère à la taille de l'élément, et non à la taille de l'image d'arrière-plan elle-même. Une valeur de 100 % couvre entièrement le fond. Dans notre cas, l'image du personnage est affichée 60 % plus grande que la largeur de la boîte et l'image de texture est de 5 % de la taille de l'élément.

Maintenant, il nous reste à modifier le positionnement du personnage (figure 3.62) :

```
#graphikBloc{
	...
	background-position: top left, top left, 20% 0, top left;
}
```

Fig. 3.62 Notre personnage est positionné

Continuons avec les effets de lumières (figure 3.63). Pour cela, nous allons ajouter quatre ombres :

- une ombre claire sur le haut de la boîte ;
- une ombre claire sur la partie interne du bas de la boîte ;
- une ombre foncée sur le bas de la boîte ;
- une ombre derrière la boîte.

```
#graphikBloc{
	...
	box-shadow: 0 -1px 0 rgba(255,255,255,0.3),
		0 -1px 0 rgba(255,255,255,0.3) inset,
		0 1px 0 black,
		0 80px 10px -50px rgba(0,0,0,.5);
}
```

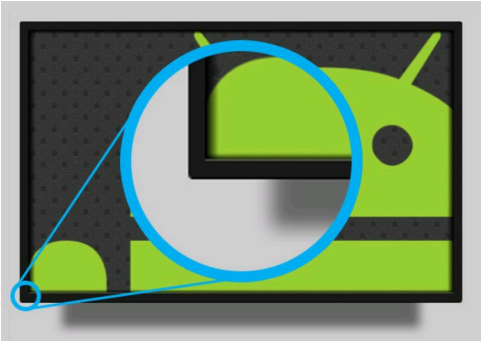


Fig. 3.63 Les ombres sont ajoutées

Note : sur la figure 3.63, la couleur #CCC a été appliquée sur l'élément parent pour accentuer les effets d'ombres.

La deuxième ombre est une ombre interne, par l'ajout du mot-clé inset. La dernière utilise une valeur d'étendue de -50 px, ce qui signifie que l'ombre créée est plus petite que la taille de l'élément.

Pour terminer, nous allons créer un nouveau dégradé, qui va simuler un reflet (figure 3.64). Pour ce faire, ajoutons le dégradé comme troisième dégradé d'arrière-plan du bloc, et modifions les propriétés adéquates :

```
#graphikBloc{
	background: linear-gradient( left, black, transparent 1.5%, transparent 98.5%, black),
		linear-gradient( black, transparent 1.5%, transparent 98.5%, black),
		 linear-gradient( 15deg, rgba(255,255,255,0) 55%, rgba(255,255,255,.2) 55%, rgba(255
		url('bugdroid.svg'),
		url('etoiles.png');
	background-size: auto, auto, auto, 160%, 5%;
	background-position: top left, top left, top left, 20% 0, top left;
}
```

Fig. 3.64 Un reflet est ajouté

Notre reflet est donc positionné entre le personnage et les dégradés créant les ombres. Le dégradé utilisé suit un angle de 15 °, et utilise trois opacités de blanc différentes.

Et voilà, c'est fini ! Non, pas tout à fait... Ce que nous venons de réaliser ne fonctionne pas sur des navigateurs trop anciens car certaines fonctionnalités utilisées ne sont pas reconnues. C'est le cas de l'image SVG, des arrière-plans multiples et de ses propriétés liées, ou encore des dégradés.

Pour gérer tous ces cas, nous allons simplement partir de la fonctionnalité la moins supportée afin de prévoir notre dégradation gracieuse. C'est le cas de l'image SVG utilisée depuis background-image. Nous décidons donc que si le navigateur ne supporte pas cet aspect, nous affichons une image

« classique » qui contient la texture et le personnage, cette image étant alignée en haut à droite (figure 3.65). (Bien entendu, ce choix est personnel et une dégradation gracieuse différente et plus précise peut être mise en place.)

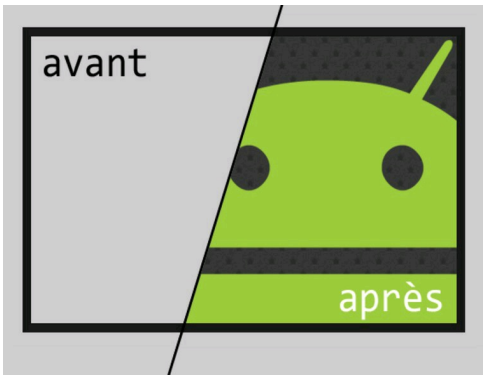


Fig. 3.65 Résultat de la dégradation gracieuse sur un navigateur très ancien

Nous déplaçons donc notre arrière-plan multiple au sein d'un nouveau bloc de style et nous ajoutons le nouvel arrière-plan au bloc initial. Le nouveau bloc ne sera lu que par les navigateurs supportant le SVG, grâce à la détection de Modernizr qui ajoute la classe .inlinesvg sur la balise html.

```
#graphikBloc{
        ...
        background: url('img/bugdroid_texture.png') top right #333;
        ...
}
.inlinesvg #graphikBloc{
        background: linear-gradient( left, black, transparent 1.5%, transparent 98.5%, black),
                    linear-gradient( black, transparent 1.5%, transparent 98.5%, black),
                     linear-gradient( 15deg, rgba(255,255,255,0) 55%, rgba(255,255,255,.2) 55%, rgba(255
                    url('bugdroid.svg'),
                    url('etoiles.png');
        background-size: auto, auto, auto, 160%, 5%;
        background-position: top left, top left, top left, 20% 0, top left;
}
```

Pour aller plus loin, nous détaillons l'animation d'introduction de ce bloc dans le chapitre dédié aux transitions et animations CSS.

Ce qu'il faut retenir...

Nous voici arrivés à la fin du chapitre consacré aux effets graphiques. Comme nous avons pu le voir grâce aux exemples concrets, CSS3 apporte de puissants « outils » pour reproduire des effets jusque-là réservés aux logiciels graphiques. Que ce soit avec la gestion de la transparence, l'ajout d'ombres ou de coins arrondis, la gestion des images d'arrière-plan ou encore les dégradés, les développeurs peuvent maintenant réaliser des mises en forme complexes et visuellement agréables, le tout en CSS. De plus, la dégradation gracieuse de ces effets est assez simple à mettre en place, voire optionnelle.

L'avenir de CSS nous promet encore de belles choses en termes d'effets graphiques, avec notamment l'arrivée des filtres CSS qui vont nous faciliter la création de flou, de filtre de netteté, de gestion du contraste ou de la luminosité des images. Nous abordons cet aspect brièvement dans le dernier chapitre.

Nous allons maintenant nous intéresser à un autre aspect graphique souvent négligé, la typographie.

Polices et typographie

Objectifs

La typographie est sans doute l'élément le plus important du design. Une attention particulière doit être apportée quant à son utilisation sur internet, qui n'est pas un média figé comme l'est un texte sur une feuille blanche. Le web est un média hétérogène, où l'affichage des données est dépendant de la configuration de l'utilisateur, où le contrôle de la taille des textes n'est pas réellement possible et où le rendu des polices est tributaire du système. CSS nous permet de minimiser ces risques par le biais de propriétés dédiées. Les nouvelles propriétés CSS3 nous promettent un contrôle encore plus poussé de la gestion des fontes sur le web. Pour ce faire, plusieurs modules sont à l'état de brouillon, c'est le cas notamment du CSS Fonts Level 3, du CSS Text Level 3, du CSS Writing Modes Level 3 ou encore du CSS Line Layout Level 3, même si ce dernier est totalement délaissé.

L'objectif de ce chapitre est d'offrir une vision détaillée de l'utilisation actuelle et future de la typographie pour le média web.

4.1. Les propriétés typographiques

Avant de voir ce qu'apporte CSS3, faisons un petit état des lieux sur l'utilisation courante des polices de caractères sur le web. Nous avons la possibilité de choisir une famille de police à utiliser pour chaque élément de notre page. En lien avec cela, il nous est possible de modifier la taille, la graisse, le type... Cela passe par l'utilisation de plusieurs propriétés CSS.

4.1.1. font-family

Cette propriété permet de spécifier la famille police que nous souhaitons utiliser pour le rendu de nos textes. Dans les faits, une liste de polices séparées par des virgules doit est précisée. La première de la liste étant le choix principal, la dernière étant généralement la famille générique de police souhaitée. Pourquoi un choix principal ? Tout simplement car une fonte ne peut pas être visualisée par l'internaute si celle-ci n'est pas installée sur son ordinateur.

Il convient alors de spécifier la règle font-family, comme ceci :

Utilisation de la propriété font-family

```
body{
	font-family: Helvetica, Verdana, sans-serif;
}
```

Dans cet exemple, si la police Helvetica n'est pas disponible chez l'utilisateur, Verdana sera choisie. Si Verdana n'est pas présente, alors le rendu sera effectué avec la police générique de type sans-serif du système. Voici les différentes familles de polices génériques dont dispose tout système(figure 4.1) :

- serif : polices à empattements ;

- sans-serif : polices sans empattements ;
- monospace : polices à chasse fixe ;
- cursive : polices manuscrites ;
- fantasy : polices décoratives.

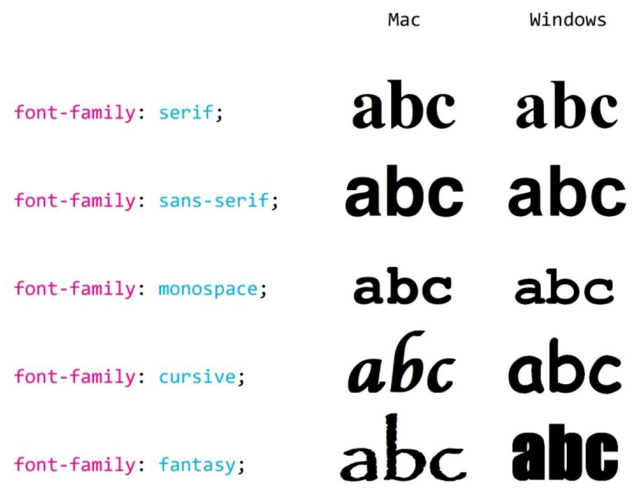


Fig. 4.1 Familles de polices génériques
sur Mac (1ère colonne) et Windows (2nde colonne)

Le site CSS Font Stack nous donne une estimation du support des polices en fonction des différents OS. De plus, le site nous permet d'obtenir la règle font-family « optimale » pour une police choisie. Par exemple, si votre choix de typo se porte sur la police Garamond, voici leur préconisation : font-family: Garamond, Baskerville, "Baskerville Old Face", "Hoefler Text", "Times New Roman", serif; Pas mal, non ?

URL : http://cssfontstack.com/

4.1.2. font-size

Cette propriété permet de définir la taille de la police.

Utilisation de la propriété font-size

```
body{
	font-family: Helvetica, Verdana, sans-serif;
	font-size: 1.1em;
}
```

Il existe deux types d'unités : les unités absolues (px, pt, cm...), ou les unités relatives (%, em ...). Les unités relatives présentent plusieurs avantages :

- elles sont agrandies si l'utilisateur utilise le zoom texte de son navigateur (sur certaines versions d'IE notamment) ;
- elles se basent sur les préférences de l'utilisateur (figure 4.2). Cela signifie que si le navigateur est configuré avec un corps de texte standard en 18 px, alors 1 em sera égal à 18 px. Utiliser une taille de police inférieure à 1 em diminue la taille des textes, tandis qu'utiliser une taille supérieure les augmente.

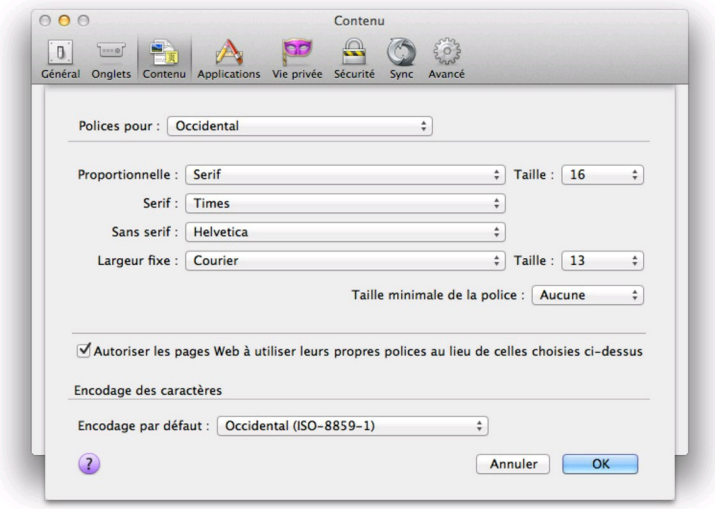


Fig. 4.2 Fenêtre Firefox de réglages des tailles de polices

La spécification CSS3 définit une nouvelle unité pour les textes, le rem. Cette unité relative permet de baser la taille d'un texte sur la taille de l'élement racine (*root* en anglais) de la page, dans notre cas html, au lieu de se baser sur la taille du parent, comme le fait l'unité em.

Prenons l'exemple d'une configuration classique où le texte par défaut est affiché en 16 px. Et utilisons le code CSS suivant :

```
div{
	font-size: 1.2em;
}
div h1{
	font-size: 2.4em;
}
```

Les textes contenus dans les div auront une taille de 16 px × 1,2 = 19,2 px et les h1 auront une taille de 19,2 px × 2,4 = 46,08 px! En utilisant l'unité rem, voici la différence :

```
div{
	font-size: 1.2em;
}
div h1{
	font-size: 2.4rem;
}
```

Les textes contenus dans les div auront une taille équivalente soit 19,2 px, par contre les h1 auront une taille de 16 px × 2,4 = 38,4 px!

rem	9.0	3.6	6.0	5.0	11.6	4.0	12.0	2.1

Fig. 4.3 Tableau de compatibilité de l'unité rem

4.1.3. font-weight, font-style, font-variant

- font-weight permet de définir la graisse de la police. Les mots clés sont normal, bold, bolder, lighter ainsi que les valeurs de 100 à 900 (avec un pas de 100), si les polices existent en différentes

graisses.

- font-style permet de définir si l'affichage doit être effectué en italique. Le mot clé est italic.
- font-variant permet de définir si l'affichage doit être effectué en petites majuscules. Le mot-clé est small-caps.

4.1.4. font

La propriété raccourcie font permet de spécifier les différentes valeurs des propriétés précédentes. Il est donc possible de définir la famille de police, la taille, la graisse, le style et également l'interlignage:

Propriété raccourcie font

```
div{
	font: bold italic 1em/1.4em Arial, sans-serif;
}
```

Dans cet exemple, les textes seront écrits en Arial gras italique, de taille 1 em et avec un interlignage de 1,4 em.

4.1.5. Propriétés de texte

En lien avec les propriétés de polices, la recommandation actuelle (CSS2.1) définit des règles CSS concernant les textes. Les voici par ordre alphabétique :

- letter-spacing définit l'écart inter-lettres.
- line-height définit l'interligne d'un bloc de texte.
- text-align définit l'alignement horizontal des éléments *inline* au sein d'un bloc
- text-decoration définit les « décorations » applicables aux textes, comme le soulignement.
- text-indent définit le décalage de première ligne d'un texte.
- text-transform modifie la casse du texte (minuscule, majuscule...).
- white-space permets de préciser la façon dont les espaces sont gérés.
- word-spacing définit l'écart inter-mots.

4.1.6. Limitations

Comme nous venons de le voir dans ce bref état des lieux, la principale limitation quant à l'utilisation des polices sur internet est liée à leurs présences chez l'utilisateur. En clair, il n'est pas possible d'utiliser une police « exotique », c'est-à-dire, une police autre que celles installées par défaut avec le système d'exploitation. Le choix étant alors très limité, on se retrouve souvent à devoir employer les mêmes familles de polices.

Pour contourner ce problème, certains concepteurs web créent des images typographiques (figure 4.4). Ces images, créées à l'aide d'un logiciel graphique, sont appliquées en CSS avec background-image et viennent en remplacement du contenu textuel. Le manque de choix de police est alors résolu ! C'est vrai. Toutefois, cette technique implique d'avoir de bonnes connaissances en termes d'accessibilité, notamment dans la manière de « cacher » le texte. Et oui, si votre texte reste présent, il apparaîtra deux fois, une fois en texte « réel » sélectionnable, et une fois sur l'image. Plusieurs solutions tentant de résoudre cet épineux problème sont proposées sur internet, sans réelle solution. Citons parmi les plus employées : text-indent négatif, padding-top avec height:0 et overflow:hidden,

ou encore l'utilisation de la balise img et de l'attribut alt en lieu et place de background-image. Malgré cela, aucune solution n'est assez convaincante en termes d'accessibilité.

Fig. 4.4 Exemple d'utilisation d'images typographiques sur www.laposte.fr

Mais ne nous inquiétons pas, car avec CSS3, les images typographiques, c'est fini ! Ou presque. En tout cas, les choses évoluent dans le bon sens et il devient possible d'utiliser n'importe quelle fonte pour son site web, même si celle-ci n'est pas présente chez l'utilisateur.

4.2. Les polices embarquées

Utiliser des fontes originales sur le web, cette possibilité a été pensée depuis longtemps, c'est le rôle de la règle @font-face. Elle est apparue au sein de CSS2 et était même implémentée dans Internet Explorer 4. Malheureusement, un format propriétaire unique, une interprétation différente des spécifications ainsi que le problème de gestion de propriété intellectuelle l'ont fait tomber aux oubliettes. C'est pourquoi elle a été retirée de CSS2.1. Mais, la voilà de retour en CSS3 et les navigateurs ont pris le train en marche !

4.2.1. @font-face

Cette règle permet de définir une police qui sera téléchargée par le navigateur afin d'être utilisée pour le rendu typographique des éléments.

Déclaration de base pour @font-face

```
@font-face{
        font-family: maPolice;
        src: url('chemin/maPolice.ttf') format('ttf');
}
```

Cette déclaration, à ajouter directement au sein de nos CSS, est définie par une @-règle. Elle est composée de deux propriétés au minimum :

- font-family permet de nommer la police. C'est le nom qui sera utilisé ensuite pour l'appliquer aux textes.
- src permet de spécifier l'adresse de la fonte à télécharger. Le format de la police est également précisé, à l'aide de la fonction du même nom.

Attention, si le format n'est pas précisé, les navigateurs devront télécharger la police avant de savoir s'ils peuvent l'utiliser ou pas. Préciser le format permet donc une optimisation des chargements puisque les navigateurs ne téléchargent pas une police si le format n'est pas supporté.

Maintenant que notre police est définie, il ne nous reste plus qu'à l'utiliser :

Utilisation d'une police définie avec @font-face

```
h1{
	font-family: maPolice, sans-serif;
}
```

Dans ce cas simple, « maPolice » est appliquée sur le h1 (figure 4.5). Le type de famille générique sans-serif est tout de même ajouté en fin de chaîne afin de gérer le cas où la fonte ne pourrait pas être affichée (problème de téléchargement, lenteur du réseau, navigateur ne prenant pas en charge @font-face). C'est également cette police qui est vue lors d'un FOUC[1]

Fig. 4.5 Utilisation de @font-face avec la police Josefin Slab

4.2.2. Compatibilité

Aujourd'hui, la majeure partie des navigateurs web implémentent @font-face. Cependant, si nous reprenons notre exemple initial, la police utilisée est au format TTF (*TrueType Font*), et ce format n'est pas interprété partout, il faut donc proposer des formats de polices différents pour un support au sein de chaque navigateur.

Il existe actuellement cinq formats de fontes disponibles pour le web. Ces formats sont :

- SVG/SVGZ (*Scalable Vector Graphics*)
- TTF (*TrueType Font*)
- OTF (*OpenType Font*)
- EOT (*Embedded OpenType*)
- WOFF (*Web Open Font Format*)

Ce dernier format, qui est le plus récent en date, est un format ouvert, créé à l'initiative d'Opéra, de Microsoft et de Mozilla, et a été spécialement pensé et conçu pour le web. Il a donc vocation à être utilisé massivement, son support devenant de plus en plus important. WOFF est à présent supporté dans la version la plus récente de chaque navigateur.

La figure 4.6 répertorie les formats de polices compatibles par navigateur.

@font-face	5.5	3.5	4.0	3.2	10.0	3.2	10.0	2.2
SVG			4.0	3.2	9.0	3.2	10.0	3.0
TTF/OTF	9.0	3.5	4.0	3.1	10.0	4.2	10.0	2.2
EOT	6.0							
WOFF	9.0	3.6	5.0	5.1	11.1	5.0	11.0	

Fig. 4.6 Tableau de compatibilité de @font-face et des formats de polices

Note : pour en savoir plus sur les formats de fichiers, nous vous conseillons la lecture de cet article :
URL : http://typographisme.net/post/Les-formats-de-polices-typographiques-pour-le-Web

4.2.3. Déclaration complète

Pour un support maximal de @font-face, notre déclaration précédente doit alors s'enrichir pour faire appel aux différents formats vus précédemment. Pour cela, nous séparons les formats de fontes par une virgule (,) au sein de la règle src en utilisant le principe classique de CSS.

De plus, il est préférable de faire appel en premier lieu au format WOFF, puisque lors du parcours de cette règle par le navigateur, si le format de la première police trouvée est reconnu, la fonte est téléchargée, et le reste de la déclaration de la règle src abandonnée. Le format WOFF est le format recommandé pour @font-face.

Déclaration pour @font-face, avec plusieurs formats de polices

```
@font-face{
	font-family: maPolice;
	src: url('chemin/maPolice.woff') format('woff'),
		url('chemin/maPolice.ttf') format('ttf'),
		url('chemin/maPolice.svg#ancre') format('svg');
}
```

Dans cet exemple, nous remarquons que toutes les versions de navigateurs trouveront une police qui leur convienne, que ce soit le format WOFF, TTF ou encore SVG sur iOS < 5. Dans le cas d'Internet Explorer 8 et inférieurs, ces formats ne suffisent pas.

4.2.4. IE le contradicteur

Bien que le support de @font-face sur Internet Explorer soit effectif depuis plusieurs versions, il demeure encore de nombreuses limitations et/ou dysfonctionnements dans les versions 6 à 8. Mais rassurons-nous, ces problèmes peuvent êtres contournés afin d'offrir un support équivalent sur toutes les versions d'IE encore utilisées. Et surtout, ces mésaventures sont oubliées depuis la version 9.

Le format .eot

La première limitation vient du format de la police supportée. En effet, les versions d'IE 6 à 8 interprètent exclusivement le format EOT, qui d'ailleurs n'est reconnu dans aucun autre navigateur.

Pour contrer cela, il suffirait d'ajouter l'appel de la police au format EOT dans la liste des polices depuis la règle src. Facile ! Sauf qu'IE souffre d'un autre problème...

Le parsage de src

Le décodage de la chaîne de caractère, ce que l'on appelle le parsage, contenue dans la règle src pose problème. En effet, le fait de séparer les différents appels par des virgules, ainsi que l'utilisation de format() sont mal interprétés. Ainsi, si nous ajoutons l'appel au fichier EOT en premier dans notre exemple précédent, comme ceci :

Déclaration pour @font-face incompatible IE < 9

```
@font-face{
	font-family: maPolice;
	src: url('chemin/maPolice.eot') format('embedded-opentype'),
		url('chemin/maPolice.woff') format('woff'),
		url('chemin/maPolice.ttf') format('ttf'),
		url('chemin/maPolice.svg#ancre') format('svg');
```

```
}
```

Internet Explorer va renvoyer une erreur 404. Cette erreur provient du fait qu'il ne détecte pas la première parenthèse fermante de la chaîne contenue dans src. Il continue sa lecture jusqu'à trouver la dernière parenthèse fermante, et interprète ce qu'il a trouvé comme un nom de fichier. Dans notre cas, le fichier qu'IE souhaite trouver s'appelle :

```
maPolice.eot')%20format('embedded-opentype'), url('chemin/maPolice.woff')%20format('woff'), url('chemin/maPolice.ttf')
```

Et donc logiquement, une erreur est renvoyée.

Une solution pour corriger ce problème pourrait être de faire appel au fichier EOT de manière unique, puis dans une seconde règle src, de faire appel aux autres formats de polices tout en restant dans la même déclaration @font-face.

Malgré cela, des développeurs soucieux d'écrire une seule règle src, ont mis au point une solution pour s'affranchir de la double définition.

4.2.5. La solution : la syntaxe Fontspring

Cette solution de contournement du problème d'IE, mise au point par Ethan Dunham[2], est toute simple, encore fallait-il y penser. L'astuce consiste à placer un point d'interrogation (?) juste après le nom du fichier EOT, ce qui permet de faire croire à IE que ce qui suit est une *query string*[3]. IE comprend donc le nom du fichier et ignore ce qui suit le point d'interrogation (?). Ce fichier est alors téléchargé sans aucun souci. La syntaxe finale est donc la suivante :

Déclaration finale pour @font-face

```
@font-face{
        font-family: maPolice;
        src: url('chemin/maPolice.eot?') format('embedded-opentype'),
             url('chemin/maPolice.woff') format('woff'),
             url('chemin/maPolice.ttf') format('ttf'),
             url('chemin/maPolice.svg#ancre') format('svg');
}
```

Attention, le type de format d'une police EOT n'est pas eot, mais bien embedded-opentype comme définit dans cet exemple.

Cette écriture est suffisante pour notre dégradation gracieuse. Cependant, afin de profiter du support d'IE9 avec la police au format WOFF, il est possible de lui faire « sauter » l'appel au format EOT. Pour cela, il nous suffit de modifier le nom du format de la police par autre chose que son format réel, et ainsi IE9 comprendra que ce format n'est pas reconnu. Il testera donc le suivant, le format WOFF, et le téléchargera. Nous pouvons donc éventuellement modifier la première ligne de notre déclaration par format('eot') ou format('ce-n-est-pas-un-format').

Un dernier problème peut se produire lorsqu'IE9 est utilisé en mode de compatibilité. Pour pallier cela, une balise meta empêche le navigateur de basculer dans ce mode et cela ne pose aucun souci sur des sites construits sur le respect des standards. Il s'agit de la meta X-UA-Compatible à ajouter au sein de la balise head.

Balise meta X-UA-Compatible

```
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
```

4.2.6. Variantes

Il est possible de forcer le navigateur à utiliser les polices de substitution adéquate pour les différentes variantes d'une police. En effet, si rien n'est précisé, les navigateurs vont simuler le gras ou l'italique sur les textes utilisant @font-face et cela peut parfois provoquer un résultat désastreux. C'est ce que l'on appelle le **faux-gras** ou le **faux-italique** (figure 4.7).

Fig. 4.7 Différence entre le gras et le faux-gras à taille de police équivalente

Pour contrer ce problème, il est possible de télécharger plusieurs types d'une même police par l'ajout d'une autre déclaration @font-face, et en précisant au sein de celle-ci les règles font-weight, font-style ou même font-variant. Voici un exemple de CSS pour une police disponible en normal et en gras :

Double déclaration pour forcer les variantes

```
@font-face{
	font-family: maPolice;
	src: url('chemin/maPolice.eot?') format('embedded-opentype'),
		url('chemin/maPolice.woff') format('woff'),
		url('chemin/maPolice.ttf') format('ttf'),
		url('chemin/maPolice.svg#ancre') format('svg');
	font-weight: normal;
	font-style: normal;
}
@font-face{
	font-family: maPolice;
	src: url('chemin/maPoliceGras.eot?') format('embedded-opentype'),
		url('chemin/maPoliceGras.woff') format('woff'),
		url('chemin/maPoliceGras.ttf') format('ttf'),
		url('chemin/maPoliceGras.svg#ancre') format('svg');
	font-weight: bold;
	font-style: normal;
}
```

Ensuite, l'appel à *maPolice* en précisant font-weight: bold force le navigateur à récupérer la version gras de la police, au lieu de le simuler.

```
h1{
	font-family: maPolice, sans-serif;
	font-weight: bold;
}
```

Cette fonctionnalité n'est malheureusement pas interprétée dans certaines versions de Chrome, de Safari et d'Internet Explorer. Les textes seront alors rendus en faux gras.

4.3. Sites de fontes

Ce que nous venons de voir, c'est que l'utilisation de polices personnalisées nécessite de prévoir plusieurs formats, ce qui peut être un frein dans l'adoption sur un site web. Nous n'avons pas tous la compétence pour créer des fontes, mais surtout, nous ne voulons pas nuire à la propriété intellectuelle d'une police. C'est pourquoi des sites en ligne proposent des bibliothèques de fontes, rapides et faciles à utiliser, mais surtout optimisées pour le web.

Ces sites sont nombreux, mais nous allons nous intéresser plus principalement à deux d'entre eux, qui proposent des polices libres de droits : **Google web fonts** et **Fontsquirrel**.

L'objectif de ces sites est d'offrir le même service, à savoir proposer des fontes pour une utilisation via @font-face, mais les méthodes de fonctionnements diffèrent l'une de l'autre.

Google web fonts part du principe que la police n'a pas à être hébergée sur chaque site, mais qu'elle doit plutôt se trouver sur leurs serveurs. C'est pourquoi, lors de l'appel du code généré par Google, celui-ci se charge de vérifier quel format de fonte est adapté au navigateur afin de ne faire télécharger que ce fichier par l'utilisateur.

Fontsquirrel voit les choses différemment. De leur côté, le téléchargement d'un pack contenant les formats de polices, et le CSS associé, est proposé au format ZIP. À charge ensuite de déployer ces polices à notre gré.

4.3.1. Google web fonts

Google web fonts nous promet des centaines de polices, libres de droits et optimisées pour le web (figure 4.8). C'est vrai. À l'heure actuelle, le site en propose plus de 400. Le choix d'une, ou de plusieurs polices, est décomposé en trois étapes :

- Choisir ses polices
- Comparer et vérifier le contenu
- Obtenir le code

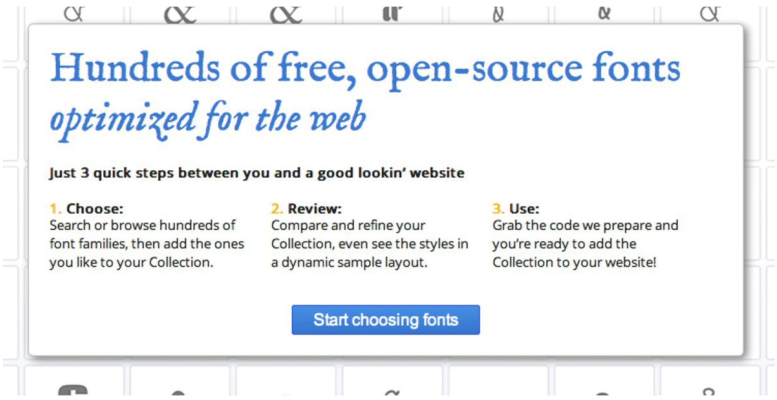


Fig. 4.8 Google web fonts nous propose des centaines de polices

Les deux premières étapes sont très classiques (choix des polices, modification de la taille, des propriétés de textes, comparaison de plusieurs polices…). La dernière étape quant à elle, nous indique notamment le temps de chargement de chaque police (figure 4.9), ce qui nous permet de détecter si un fichier est anormalement lourd ou si une police similaire ne serait pas plus légère.

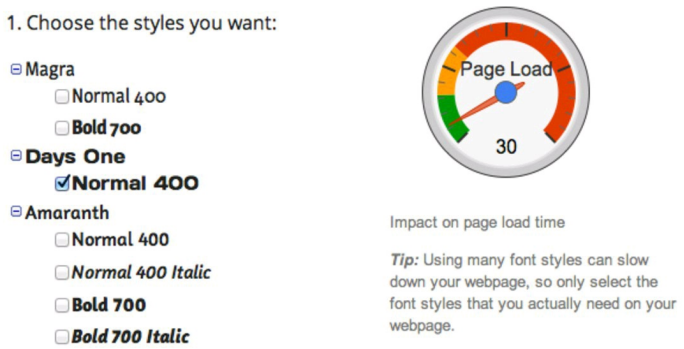


Fig. 4.9 Temps de chargement de la police sélectionnée

Ensuite, le site nous laisse choisir la méthode d'import de nos polices :

- *via* une balise link
- *via* la règle @import
- *via* JavaScript

Exemple d'import de la police via la balise link

```
<link href="http://fonts.googleapis.com/css?family=Days+One" rel="stylesheet" type="text/css">
```

Le code généré par cet appel est alors spécifique au navigateur du client. Dans cette déclaration, seul un format de fonte est présent, celui que Google a considéré comme le plus adapté pour un navigateur donné. La fonction local() permet de vérifier si la police est déjà installée chez l'utilisateur. Si c'est le cas, elle n'est pas téléchargée.

Exemple de CSS généré pour Chrome

```
@font-face{
        font-family: 'Days One';
        font-style: normal;
        font-weight: 400;
        src: local('Days One'),
             local('DaysOne-Regular'),
             url('http://chemin/xxx.woff') format('woff');
}
```

L'utilisation est alors classique :

Utilisation de la police choisie sur les titres

```
h1, h2, h3{
        font-family: 'Days One', sans-serif;
}
```

4.3.2. Fontsquirrel

Fontsquirrel propose également des centaines de polices (figure 4.10), mais contrairement à *Google web fonts*, elles se trouvent sous forme de kit à télécharger. Ces kits sont multiples et sont classés par catégories de polices (police à empattements ou non, à chasse fixe, manuscrite...).

Fig. 4.10 Polices téléchargeables sur FontSquirrel

@font-face Kit

Les kits (figure 4.11)sont pourvus des formats de fontes suivants pour être compatible dans chaque navigateur :

- TTF et OTF
- EOT
- SVG
- WOFF

Ainsi que deux fichiers qui permettent de copier-coller rapidement la syntaxe de @font-face dans nos pages, ou pour effectuer nos tests : demo.html et stylesheet.css.

Par défaut, les kits sont téléchargés avec le jeu de caractères *MacRoman*, c'est-à-dire avec la gestion des accents (ou d'autres caractères). Ce qui nous convient parfaitement.

Il est toutefois possible de télécharger un jeu de caractères différents, par exemple pour une page en anglais, qui ne contiendrait pas d'accent. Les fichiers de fontes sont alors plus légers.

Il est également possible d'ôter certains formats du téléchargement.

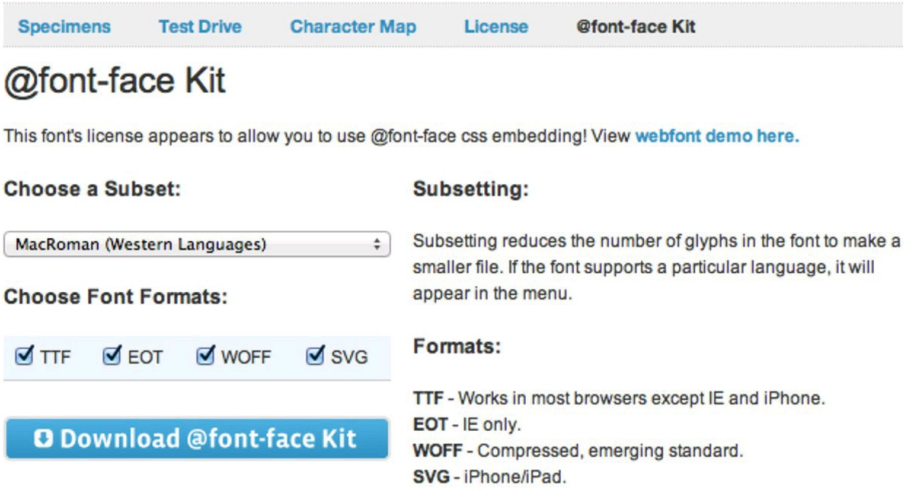


Fig. 4.11 Récupération du kit @font-face

@font-face Generator

Cette option nous propose d'aller plus loin avec @font-face, en nous donnant la possibilité de créer nos propres kits à partir d'une police de notre création, ou d'une police libre de droits.

Pour cela, un formulaire est proposé dans lequel nous devons tout d'abord sélectionner notre police puis définir le mode de création (figure 4.12).

Le mode expert nous permet un contrôle très poussé sur la façon dont notre kit va être créé. Les différentes options applicables sont expliquées sous le formulaire. Citons, par exemple, la possibilité de choisir les formats de polices créés, la syntaxe @font-face à utiliser, les jeux de caractères utilisées ainsi que pleins d'autres options avancées. Au final, nous obtenons un kit prêt à l'emploi pour notre police.

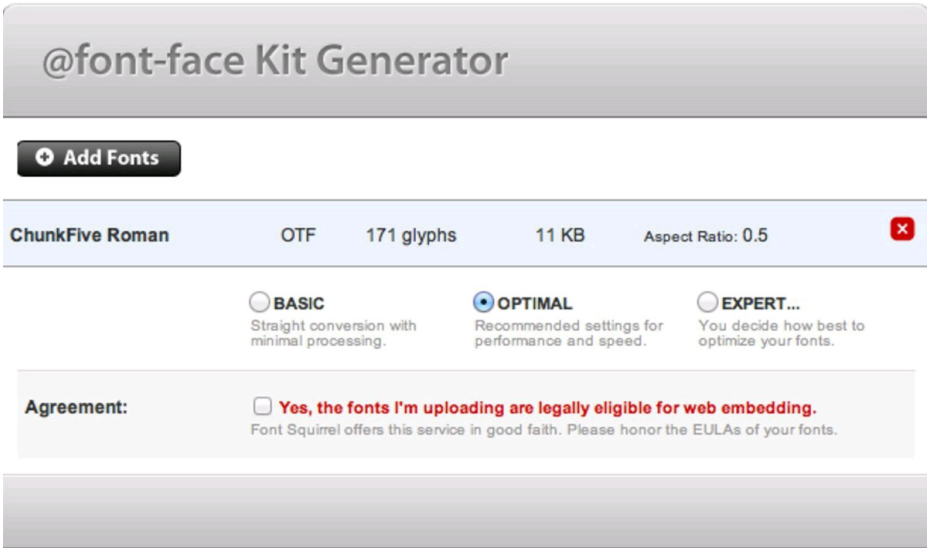


Fig. 4.12 Génération d'un kit depuis une police

4.4. Les nouvelles propriétés

Les nouveaux modules CSS3, encore à l'état de réflexion, étendent le nombre de propriétés liées à la gestion de la typographie. L'objectif premier étant d'avoir un plus grand contrôle concernant le rendu

de la typographie elle-même, les lettres, les mots, les espaces blancs, les retours à la ligne, les alignements, les débordements et en règle générale, tout ce qui se rapporte aux textes.

Parcourons les deux modules principaux de ce domaine, le *CSS Fonts Level 3* et le *CSS Text Level 3*.

> **Note** : bien que les propriétés décrites ci-dessous soient considérées comme nouvelles, il faut savoir que certaines d'entre elles étaient présentes depuis la spécification CSS2 qui n'a pas vu le jour, qu'elles ont étés retirées de CSS2.1, et qu'elles sont de nouveau à l'étude dans CSS3.

4.4.1. Les propriétés de polices

font-stretch

La propriété font-stretch permet, comme son nom l'indique, de condenser ou à l'inverse, d'étirer les caractères d'une police. Le W3C définit neuf mots-clés allant du plus condensé au plus étiré :

- ultra-condensed
- extra-condensed
- condensed
- semi-condensed
- normal
- semi-expanded
- expanded
- extra-expanded
- ultra-expanded

Ces valeurs sont applicables uniquement sur une famille de polices dont les styles condensés ou étirés existent. En clair, si aucune variante d'étirement n'existe au sein de la fonte, aucun de ces mots-clés ne fonctionnera. En revanche, prenons l'exemple d'une famille de police qui aurait deux styles: normal et condensé. Dans ce cas, le style condensé sera utilisé pour les valeurs ultra-condensed à semi-condensed, et le style normal sera utilisé de normal à ultra-expanded. C'est le cas notamment de la police Myriad Pro sur Mac.

font-stretch font-stretch

Fig. 4.13 Résultat de font-stretch en condensé et en normal

font-stretch	9.0	9.0						

Fig. 4.14 Tableau de compatibilité de font-stretch

font-size-adjust

La propriété font-size-adjust permet d'ajuster la *hauteur d'x* d'une police, afin d'uniformiser le rendu visuel des textes ou d'en garder la lisibilité, par exemple lorsque notre choix de police n'est pas présent chez l'utilisateur. font-size-adjust permet alors de faire correspondre la *hauteur d'x* d'une typographie par rapport à une autre (figure 4.15).

Fig. 4.15 Comparatif de la hauteur d'X
entre la police Verdana et la police Times en taille 12 px

Prenons l'exemple d'un choix de famille de police avec le CSS suivant :

```
p{
	font-family: Verdana, Myriad Pro, sans-serif;
	font-size: 12px;
}
```

Si l'utilisateur n'a pas la police Verdana, c'est la police Myriad Pro qui est utilisée pour le rendu, si celle-ci est disponible bien entendu. Ces deux polices, bien que similaire, ont une *hauteur d'x* très différente, ce qui risque de casser nos mises en page (figure 4.16).

Verdana 12px Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Myriad Pro 12px Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Fig. 4.16 Différence de hauteur d'x entre deux polices

Grâce à font-size-adjust, il suffit de forcer la *hauteur d'x* de la police Myriad Pro à utiliser la *hauteur d'x* de celle de Verdana. Chaque police à une valeur d'aspect, qui correspond à cette *hauteur d'x*. Dans le cas de Verdana, cette valeur est de 0,545. Ce qui nous donne :

Utilisation de font-size-adjust (figure 4.17)

```
p{
	font-family: Verdana, Myriad Pro, sans-serif;
	font-size: 12px;
	font-size-adjust: 0.545;
}
```

font-size-adjust: 0.545;

Verdana 12px Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Myriad Pro 12px Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Fig. 4.17 Rendu visuel de font-size-adjust

font-size-adjust		3.0						

Fig. 4.18 Tableau de compatibilité de font-size-adjust.
Malheureusement, cette propriété n'est supportée que par Firefox, depuis la version 3.

Note : un site génial pour tester vos rendus et connaître la valeur d'aspect de chacune de vos polices :
URL : http://www.brunildo.org/test/xheight.pl

font-variant-* et font-feature-settings

Les typographies OpenType (la plupart des formats actuels) sont composées d'un ensemble de caractéristiques avancées et/ou spécifiques à une langue. Elles peuvent comporter de multiples glyphes, lesquels sont différents des caractères. Par exemple, les glyphes « a », « a gras » et « a italique » sont trois glyphes différents du même caractère : l'italique n'est pas seulement un caractère penché, tout comme le gras n'est pas un caractère empâté. Les accents peuvent également figurer comme des glyphes à part entière, utilisés en complément d'autres caractères.

En suivant ce même principe, ce type de format de police permet d'utiliser les petites capitales (qui ne sont pas forcément des réductions de lettres majuscules), les ligatures (qui sont des glyphes composés de plusieurs caractères), les indices et exposants (qui ne sont pas simplement une réduction des lettres positionnées en haut ou en bas de la ligne), les fractions, les caractères non latins, etc. Ce format propose aussi une gestion intelligente des relations entre les caractères, comme par exemple le crénage (*kerning*) ou l'utilisation de caractères alternatifs, tels les paraphes ou les ligatures historiques (figure 4.19).

Fig. 4.19 Caractéristiques OpenType avancées
Exemples extraits en partie depuis le site du W3C

Le W3C nous propose d'utiliser la capacité qu'ont certaines familles de polices à pouvoir contrôler l'affichage ou non de ces glyphes et de ces caractéristiques stylistiques *via* CSS. Cela se fait à l'aide de la propriété font-variant qui a été étendue par rapport à CSS2.1 et qui fonctionne maintenant comme la notation raccourcie d'un ensemble de règles, chacune définissant une particularité :

- font-kerning active ou non le crénage des caractères (OpenType : kern) ;
- font-variant-position active la gestion des indices, exposants et ordinaux (OpenType : subs, supr, ordn) ;

- font-variant-ligatures active la gestion des différents types de ligatures, qu'elles soient classiques, optionnelles, historiques... (OpenType : liga, dlig, hlig) ;
- font-variant-caps active les petites majuscules et les petites capitales (OpenType : smcp, pcap...) ;
- font-variant-numeric permet un contrôle pour l'affichage des nombres, comme les nombres elzéviriens ou les fractions (OpenType : lnum, onum, frac...) ;
- font-variant-alternates active les glyphes alternatifs d'un même caractère, tels que des paraphes, des formes historiques, stylistiques, ou spécifiques à un langage (OpenType : swsh et cswh, hist, salt, ss01 à ss20...) ;
- font-variant-east-asian permet un contrôle de l'affichage et/ou la substitution des glyphes asiatiques (OpenType : simpl, trad, jp78...).

La propriété font-feature-settings propose un contrôle encore plus fin de la gestion de ces caractéristiques, par le biais de mots clés définis au sein du registre OpenType disponible à cette adresse :

> **URL** : http://www.microsoft.com/typography/otspec/featurelist.htm

Par exemple, pour activer les ligatures classiques au sein d'un paragraphe, voici le code CSS utilisé :

Activation des ligatures classiques (figure 4.20)

```
p{
	font: 1em Bello Pro, cursive;
	font-feature-settings: "liga" 1;
}
```

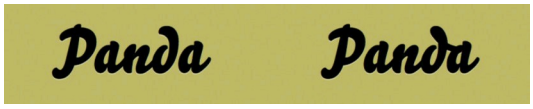


Fig. 4.20 Police Bello Pro (ligatures sur les lettres « da »).
Les navigateurs compatibles les activent par défaut.

Activation des ligatures décoratives (figure 4.21)

```
p{
	font: 1em Bello Pro, cursive;
	font-feature-settings: "dlig" 1;
}
```

Fig. 4.21 Police Bello Pro (ligatures décoratives sur les lettres « and »)

Activation des petites majuscules et des fractions (figure 4.22)

```
p{
	font: 1em Bello Pro, cursive;
	font-feature-settings: "smcp" 1, "frac" 1;
}
```

1/2 cents 1/2 CENTS

Fig. 4.22 Police Bello Pro (fractions et petites majuscules)

font-feature-settings	10.0 -ms-	4.0 -moz-	16.0 -webkit-					

Fig. 4.23 Tableau de compatibilité de font-feature-settings

Firefox ne reconnait pas la syntaxe officielle du W3C. Par exemple, pour les ligatures il faut écrire -moz-font-feature-settings: "liga=1"; Les propriétés font-variant-* telles que présentées ici ne sont toujours pas reconnues.

Visualisez les 951 glyphes de la police Bello Pro :
URL : http://new.myfonts.com/fonts/underware/bello/pro/glyphs.html

4.4.2. Les propriétés de textes

D'autres propriétés viennent s'ajouter à la gestion de la typographie sur le web, c'est le cas des propriétés de textes définies dans le *CSS Text Level 3*. Outre les propriétés connues de CSS2.1 expliquées en début de ce chapitre, ce module propose notamment de nouvelles méthodes pour contrôler les débordements au sein des blocs de textes. Voici un rapide aperçu de ces propriétés.

white-space = text-wrap + text-space-collapse

Présente dans la norme CSS2.1, la propriété white-space permet de préciser la façon dont les espaces présents dans le code source sont gérés. Les valeurs possibles de white-space sont :

- normal : c'est la valeur par défaut. Elle permet un retour à la ligne classique, et annule les espaces multiples présents dans le code source.
- pre : les multiples espaces sont affichés et des lignes sont créées uniquement si des sauts de lignes sont présents dans le code source. Sans sauts de lignes présents, le texte s'affichera sur une ligne unique.
- nowrap : les multiples espaces et les retours à la ligne sont annulés. Le texte est alors écrit sur une seule ligne.
- pre-wrap : c'est la combinaison des valeurs pre et normal. Les multiples espaces sont affichés, les retours à la ligne sont effectués s'ils sont présents dans le code, et des lignes sont créées classiquement si le texte est plus grand que son conteneur.
- pre-line : équivaut à pre-wrap sauf que les multiples espaces sont annulés.

En CSS3, deux nouvelles propriétés permettent d'affiner ce réglage : text-wrap et text-space-collapse :

- text-wrap permet de spécifier la façon dont les sauts de lignes sont gérés :
  - normal : les sauts de lignes sont effectués au niveau des points de ruptures classiques.
  - none : aucun saut de ligne n'est créé, le texte est écrit sur une ligne unique.
  - avoid : aucun saut de ligne n'est créé, sauf si des points de ruptures autres que ceux classiques sont présents. Oui, c'est assez flou...

- text-space-collapse permet de spécifier la façon dont les espaces blancs présents dans le code source sont gérés :
  - collapse : les espaces blancs multiples et les tabulations sont regroupés en un caractère espace blanc unique.
  - preserve : les espaces blancs multiples et les tabulations sont conservées, tout comme les sauts de lignes manuels.
  - preserve-breaks : les espaces blancs multiples et les tabulations sont regroupés en un caractère espace blanc unique, mais les sauts de lignes manuels sont conservés.

La propriété white-space fonctionne donc à présent comme le raccourci de ces deux propriétés, dont voici le tableau d'équivalence (tableau 4.1).

Tab. 4.1 Tableau de correspondance entre white-space, text-wrap et text-space-collapse

white-space	text-wrap	text-space-collapse
normal	normal	collapse
pre	none	preserve
nowrap	none	collapse
pre-wrap	normal	preserve
pre-line	normal	preserve-breaks

Voici les différents cas d'application de white-space, sur un paragraphe avec retour à la ligne dans le code source (figure 4.24)

```
<div>
<p>Lorem ipsum
dolor sit amet, consectetur adipiscing elit.</p>
</div>
```

Notez les espaces présents devant le mot dolor dans le code source, car cela a une influence sur le rendu final.

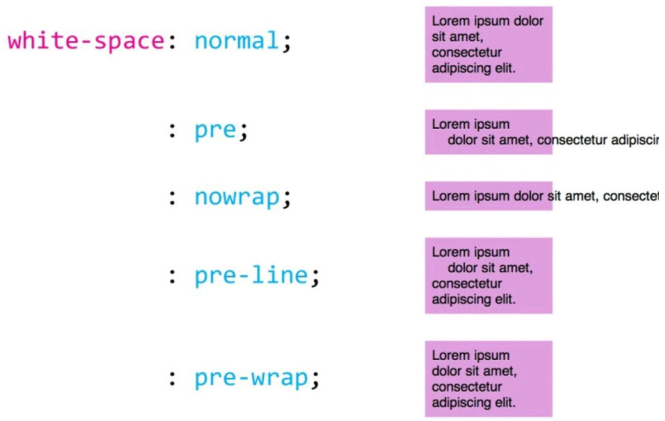


Fig. 4.24 Effets de l'application des différentes valeurs de white-space

white-space	8.0	3.5	4.0	3.1	9.5	3.2	10.0	2.1

Fig. 4.25 Tableau de compatibilité de white-space.
Le tableau fait état du support complet de white-space (cinq valeurs).
Aucun navigateur n'implémente encore les propriétés text-wrap et text-space-collapse.

text-overflow

La propriété text-overflow détermine la façon de traiter un texte lorsqu'il déborde de son parent, notamment lors de l'utilisation conjointe avec white-space (figure 4.26). Le texte doit avoir une valeur d'overflow différente de visible. Les valeurs possibles sont les suivantes :

- clip : c'est la valeur par défaut. Le texte est purement et simplement rogné aux limites de son parent. Le texte qui déborde est donc invisible.
- Ellipsis : le texte est également coupé, mais des points de suspension (...) apparaissent, afin de signaler que le texte affiché n'est pas complètement visible.

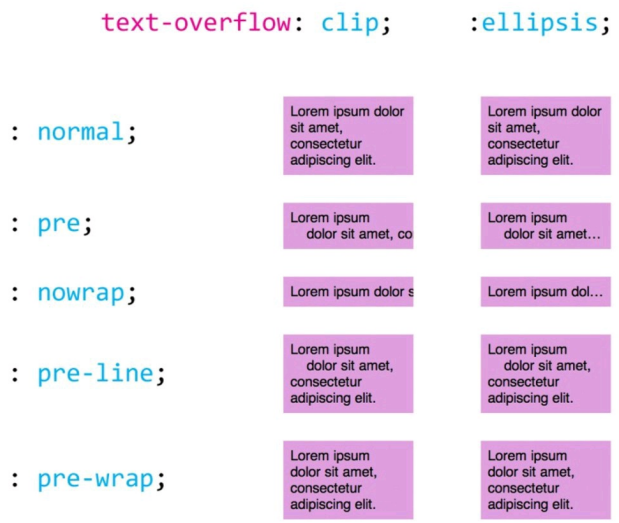


Fig. 4.26 Effets de text-overflow sur un élement en overflow : hidden; et les différentes valeurs de white-space

Attention, text-overflow ne fonctionne que si les lignes dépassent vers la droite du conteneur, et non vers le bas. Il est donc impossible d'utiliser cette règle sur un bloc de plusieurs lignes pour que les points de suspension s'affichent dès que la hauteur de l'élément parent devient trop petite.

text-overflow	6.0	7.0	4.0	3.1	9.0 -o-	11.0	3.2	10.0 -o-	2.1

Fig. 4.27 Tableau de compatibilité de text-overflow

La définition de cette propriété a été déplacée, elle se retrouve maintenant au sein du module CSS3 UI Module. De plus, l'ajout d'une nouvelle valeur est en cours de proposition, celle-ci permettrait d'utiliser une chaîne de caractères de notre choix pour remplacer les points de suspension appliqués avec ellipsis. La syntaxe est encore à définir précisément, même si un test expérimental est implémenté sous Firefox 9.

overflow-wrap = word-wrap

La propriété overflow-wrap est strictement équivalente à la propriété CSS2.1 word-wrap. Elle permet de gérer la façon dont les mots trop longs sont affichés au sein de leur parent. Pour cela, nous disposons de deux valeurs :

- normal : les mots trop longs peuvent débordent de leur conteneur si celui-ci est trop petit, sauf si overflow: hidden est spécifié.
- break-word : un saut de ligne est effectué arbitrairement au milieu des mots trop longs. Cependant, aucun trait d'union n'est ajouté, la césure n'est pas prise en charge.

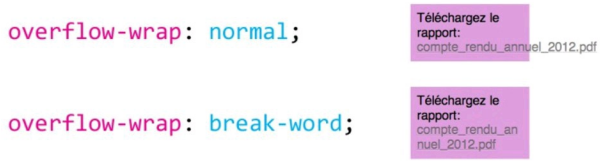


Fig. 4.28 Utilisation d'overflow-wrap

Aucun navigateur n'implémente overflow-wrap. L'utilisation de word-wrap est donc obligatoire, surtout que cette règle est supportée depuis très longtemps dans les navigateurs.

hyphens

C'est sans nul doute la règle la plus prometteuse concernant la gestion des textes. En effet, la propriété hyphens permet d'activer la césure automatique des mots en fin de ligne par l'ajout de traits d'union (dans la majorité des langues). Cette gestion intelligente de l'affichage des paragraphes permet une aide à lecture non négligeable, notamment sur des textes justifiés.

Cette règle accepte trois valeurs :

- none : pas de césure, c'est le rendu par défaut.
- auto : la césure automatique est effectuée en fonction de la langue de la page.
- Manual : la césure est effectuée seulement si un trait d'union conditionnel, tel , est présent dans un mot.

Activer la césure automatique (figure 4.29)

```
div{
	hyphens: auto;
}
```

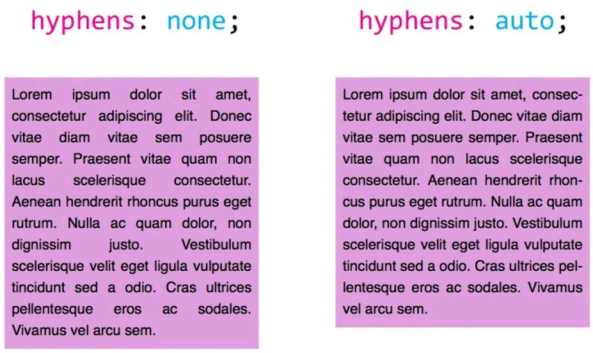


Fig. 4.29 Utilisation de hyphens sur un bloc de texte

Lorsque la césure est active, les mots trop longs sont également coupés afin d'éviter le débordement.

La règle overflow-wrap (word-wrap) devient alors secondaire (figure 4.30).

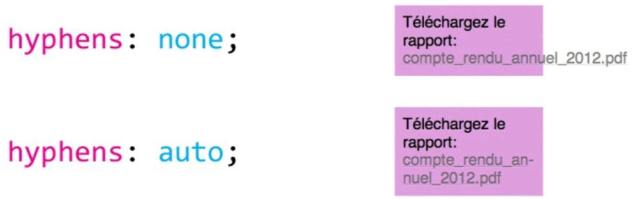


Fig. 4.30 Utilisation de hyphens sur les mots trop longs

Une gestion plus fine de la césure était prévue à l'aide de plusieurs règles CSS, mais celles-ci ont été déplacées au sein de CSS4 pour une étude plus poussée. C'est le cas de hyphenate-character, hyphenate-limit-zone, hyphenate-limit-word, hyphenate-limit-lines et hyphenate-limit-last, qui prévoient de choisir le caractère de césure, d'empêcher la césure de plusieurs lignes successives ou encore de définir le nombre minimum de lettres restantes en bout de ligne lorsqu'un mot est coupé. Certaines de ces règles sont pourtant déjà implémentées dans Internet Explorer 10.

Un espace de test est disponible à cette adresse :

> **URL** : http://ie.microsoft.com/testdrive/Graphics/hands-on-css3/hands-on_multi-column.htm

hyphens	10.0 -ms-	6.0 -moz-		5.1 -webkit-		4.2 -webkit-		

Fig. 4.31 Tableau de compatibilité de hyphens

Ce qu'il faut retenir...

Nous voici arrivés à la fin du chapitre consacré à la gestion de la typographie. Aujourd'hui, CSS3 et la règle @font-face font s'envoler les limitations d'utilisation des polices de caractères sur le web. Les développeurs et designers web peuvent enfin utiliser la police de leur choix, sans aucune restriction, puisque cette @-règle est très bien supportée.

Ce que nous promet CSS3, c'est d'aller encore plus loin dans le contrôle des débordements et des glyphes avec notamment la possibilité de profiter des caractéristiques des polices OpenType. Chaque concepteur web pourra bientôt afficher les ligatures décoratives ou historiques, les fractions, les variantes stylistiques, ou même contrôler le crénage d'une police. Cela augure de très bonnes choses, et notamment pour les langues qui demandent un contrôle poussé des signes et qui utilisent des alphabets différents du nôtre comme le thaï ou le cyrillique. D'ailleurs la spécification qui traite ce sujet est très conséquente.

Malheureusement, le support de la majeure partie des propriétés liées à la typographie reste encore très anecdotique. Cependant, la propriété font-feature-settings, dont le support a commencé depuis Chrome 16, devrait arriver prochainement sur Safari et Android qui utilise le même moteur.

Pour continuer, nous allons étudier dans le prochain chapitre les différents modes de positionnement et de gestion de la mise en page qui s'offre à nous pour construire nos pages web. Là encore, CSS3 est très attendu.

Notes

[1] FOUC (*Flash Of Unstyled Content*) : affichage des éléments non stylés lorsque la police est en cours de téléchargement.

[2] Créateur de Fontsquirrel et Fontspring.

[3] Une chaîne de caractères qui permet de passer des données de type clé/valeur dans l'URL.

Le positionnement

Objectifs

CSS3 tente de remédier aux différents problèmes de mise en pages, en nous proposant plusieurs modules dédiés qui permettent une gestion globale de placement au sein d'une page web. Pourtant, la spécification CSS2.1 définit déjà depuis plusieurs années des techniques de positionnement avancées. Celles-ci sont encore trop peu utilisées. Cela est en partie dû à IE7 et inférieurs qui ne supportent pas ces modèles. Cependant, la version 8 de ce navigateur respecte et implémente une grande majorité de ces standards, il est donc temps de s'y mettre !

L'objectif de ce chapitre est de vous offrir une vision globale de la gestion de la mise en pages.

5.1. Positionnement CSS

Avant de s'intéresser aux innovations apportées avec CSS3, revenons brièvement sur les principes actuels de la mise en page en CSS.

Il n'y a encore pas si longtemps, l'architecture d'un site web était basée sur une mise en pages construite à partir de tableaux HTML (principalement à l'aide des balises table, tr, td). Chaque élément du site se situe alors dans une cellule qui appartient à une ligne, et cela dans l'ordre d'écriture du code source (figure 5.1). Cette technique de positionnement apporte quelques avantages, comme la gestion de blocs de mêmes hauteurs, le centrage vertical, le contrôle du flux de la page... Elle apporte surtout beaucoup d'inconvénients : mise en page figée, problèmes d'accessibilité, mauvaise performance en terme de rendu, mais principalement, le HTML n'a pas vocation à gérer la mise en page. Il fallait trouver autre chose !

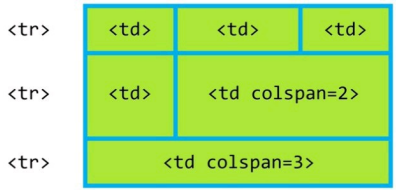


Fig. 5.1 Structure simplifiée de mise en page en tableaux HTML

Progressivement, les structures en tableaux ont alors disparu, pour laisser place à des sites construits *via* l'utilisation de balises div qui permettent un découpage de la page en blocs. Ces blocs ont par défaut une largeur égale à celle de leur parent. Cela implique qu'un positionnement horizontal des éléments n'est pas possible et qu'en conséquence, ceux-ci s'empilent les uns en dessous des autres. L'utilisation de la propriété float (et de clear pour annuler son effet) permet alors de placer des éléments côte à côte, pour peu que ces blocs soient dimensionnés. Cette propriété n'a jamais été créée pour cela, mais c'est une des techniques qui nous permet de positionner plusieurs blocs horizontalement sans l'utilisation de tableaux (figure 5.2). Cependant, cette règle n'est pas idéale : sortie de l'élément du flux, décalage non souhaité, hauteurs de blocs différentes, parent sans hauteur si tous les éléments fils sont flottants ou encore bugs des navigateurs sont autant de problèmes que nous

devons gérer au quotidien.

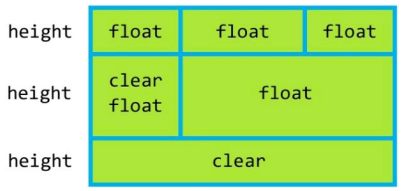


Fig. 5.2 Structure simplifiée de mise en pages par blocs flottants

Par ailleurs, la règle position, avec ses valeurs relative, absolute ou fixed nous autorise à placer un élément dans la page, à l'endroit ou on le souhaite à la manière d'un calque, sans se soucier des autres blocs ou de l'organisation du code source (figure 5.3). Malheureusement, cette solution ne nous permet pas non plus d'avoir une politique de placement globale, notamment à cause de la sortie du flux de l'élément.

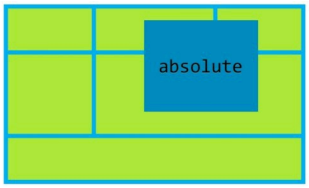


Fig. 5.3 Structure simplifiée de positionnement absolu

5.2. Modèle de boîtes

Les sites actuels reposent sur une architecture de boîtes, que ce soit des boîtes de types blocs, des textes, des tableaux ou des éléments positionnés. La spécification CSS2.1 définit ainsi quatre modèles de positionnement :

- *block layout* utilisé pour les pages web « classiques » ;
- *inline layout* utilisé pour le texte ;
- *table layout* utilisé pour des données tabulaires ;
- *positioned layout* utilisé pour le positionnement d'élément autonome au sein de la page.

Chaque balise HTML suit un modèle de positionnement par défaut. Par exemple, les p, les div, les h1-h6 respectent le modèle de type *block*. Les a, les strong, les em ou encore les img respectent le modèle de type *inline*. Les table, les tr, les td respectent le modèle tabulaire. Chaque mode de positionnement induit des normes : c'est le modèle de boîte. Revenons brièvement sur les différents modèles existants.

Le modèle de boîte de type *block* est le modèle de boîte classique (figure 5.4). Les principaux avantages de ce modèle sont que les boites de type *block* peuvent êtres dimensionnées avec les propriétés width et height, et qu'elles sont pourvues de marges internes (padding) et de marges externes (margin) qui nous permet d'espacer les boites les unes des autres. Les inconvénients majeurs sont qu'il n'existe pas d'alignement vertical du contenu au sein de la boîte, qu'il n'y a pas de lien avec le contexte (les boîtes sont autonomes), mais surtout que le calcul de la taille d'une boîte se fait en ajoutant les valeurs de largeur (width), de marge interne (padding) et de bordures (border).

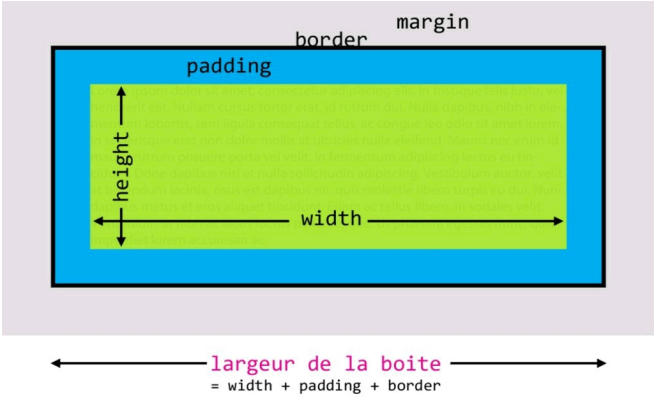


Fig. 5.4 Modèle de boîte de type block

Mode Quirks : la présence d'un DOCTYPE est indispensable au bon rendu du modèle de boîte pour Internet Explorer. En effet, sans cette information, le navigateur de Microsoft bascule dans un mode de compatibilité, appelé mode Quirks. Ce mode modifie le calcul de la taille des boîtes.

Le deuxième type de modèle de boîte, le type *inline*, ne définit pas de boîte autonome contrairement aux blocs, par conséquent les éléments s'affichent côte à côte. La boîte ne peut cependant pas être dimensionnée. En revanche, un alignement vertical des éléments *inline* contenus dans un élément de type *block* est possible grâce à la propriété vertical-align. Dernières contraintes, les marges internes ne fonctionnent que sur la droite et la gauche de l'élément, les marges externes quant à elle, ne décalent pas réellement les éléments environnants.

Le troisième modèle de boîte est le modèle tabulaire. L'avantage principal de ce modèle est que le positionnement des boîtes est dépendant du contexte, sans création de boîte autonome. C'est aussi son principal inconvénient. Ainsi deux boîtes peuvent se positionner côte à côte, leur taille peut être modifiée, et un alignement vertical est possible. De plus la largeur et la hauteur des boîtes sont « fluides » et s'adaptent aux boîtes environnantes. C'est un avantage quand il s'agit de deux boîtes côte à côte car les hauteurs seront identiques mais cela peut être un inconvénient dans le cas de plusieurs lignes, car les largeurs s'adapteront entre elles.

Ce modèle, utilisé avec les balises HTML de tableaux, a beaucoup d'autres inconvénients comme expliqué en introduction. Mais grâce à CSS et à la propriété display, nous allons pouvoir profiter de tous ses avantages.

Le dernier modèle de boîte concerne les éléments positionnés hors du flux. Les éléments HTML basculent dans ce modèle lorsque les propriétés CSS position ou float sont utilisées. Dans ce cas les éléments n'ayant plus de parent, n'ont plus de taille par défaut. Leur taille est donc dépendante de leur contenu. Pour le reste, le modèle de boîte de type *block* s'applique.

5.2.1. overflow

Pour chacun de ces modèles de boîte, la propriété overflow permet, comme son nom l'indique, de définir la méthode de dépassement des textes de leur conteneur. En effet, lorsqu'un contenu ne tient pas dans l'espace défini, celui-ci déborde. Depuis CSS2.1, cette propriété accepte quatre valeurs:

- visible : le contenu qui sort de son conteneur est visible.
- hidden : le contenu est rogné aux limites du conteneur, et aucune barre de défilement n'est ajoutée.
- scroll : le contenu est rogné aux limites du conteneur, et des barres de défilement sont ajoutées.

- auto : le choix est laissé aux navigateurs.

CSS3 étend ce mécanisme en proposant deux nouvelles propriétés, overflow-x et overflow-y. Ces propriétés, qui acceptent les mêmes valeurs, permettent de gérer le débordement pour l'axe X (horizontal) ou l'axe Y (vertical) indépendamment. Ces deux propriétés s'utilisent de manière non préfixée et sont compatibles dans la majorité des navigateurs (même IE depuis la version 6).

Un cas très pratique d'utilisation est l'ajout systématique d'une barre de défilement verticale sur l'élément html. En effet, une page dont le contenu est court n'affiche pas de barre de défilement, alors qu'une page plus longue en affiche une. La taille de la fenêtre s'en trouve alors réduite et lors du changement de page, un site centré donne l'impression d'être sans cesse décalé. La propriété overflow-y: scroll permet d'éviter ces décalages en intégrant une barre de défilement dans tous les cas.

Nous allons maintenant voir qu'il est possible de forcer un élément HTML à utiliser tel ou tel modèle de boîte, grâce à la propriété display. Nous verrons également que CSS3 introduit de nouveaux modèles. Mais tout d'abord, intéressons-nous à la propriété box-sizing qui permet d'étendre le modèle de boîte classique.

5.2.2. box-sizing

Le modèle de boîte de la spécification CSS 2.1 est construit ainsi: lorsque nous fixons la largeur ou la hauteur d'un élément HTML, comme une div par exemple, cette valeur correspond au contenu de l'élément. Cela veut dire que les bordures et les padding augmentent la taille de cette boîte. Prenons cet exemple CSS appliqué à une div :

```
div{
	width: 200px;
	border: 5px solid black;
	padding: 20px;
}
```

Nous fixons notre largeur à 200 px, puis nous ajoutons une bordure de 5 px ainsi qu'une marge interne de 20 px à l'intérieur de la boîte. Quelle sera alors la taille réelle d'affichage ? 200 px ? Non ! La taille de rendu sera de 250 px, car la *largeur d'une boîte* = width + border + padding. Si nous voulons que la boîte mesure précisément 200 px de large, il convient de modifier la règle width pour que la somme des trois propriétés soit 200 px : dans notre cas, nous devrions écrire :

```
div{
	width: 150px;
	border: 5px solid black;
	padding: 20px;
}
```

La propriété CSS3 box-sizing, définie dans le *CSS3 Basic User Interface Module*, permet alors de modifier la méthode de calcul des dimensions d'une boîte. Cette propriété accepte trois valeurs :

- content-box (modèle de boîte par défaut) ;
- border-box : les bordures et les marges internes sont comprises dans les dimensions de la boîte ;
- padding-box : seules les marges internes sont comprises.

Reprenons l'exemple précédent, mais en ajoutant le box-sizing: border-box; :

Modification du modèle de boîte

```
div{
	box-sizing: border-box;
	width: 200px;
	border: 5px solid black;
	padding: 20px;
}
```

Les bordures et les marges internes étant incluses à l'intérieur de notre élément, celui-ci mesurera donc exactement 200 px. La zone d'affichage correspondant au contenu s'en trouve alors réduite.

Cette nouvelle propriété permet surtout de s'affranchir du calcul que nous devons effectuer lorsque nous fixons la taille d'un élément qui a des bordures et/ou des marges internes. Elle devient également très pratique lorsque nous utilisons les pourcentages pour fixer la taille d'un élément, tout en ajoutant des marges internes en px ou en em.

```
div{
	box-sizing: border-box;
	width: 100%;
	padding: 1em;
}
```

Dans cet exemple (figure 5.5), la largeur de l'élément (vert) est de 100 % de la taille de son parent (bleu) qui est de 500 px. Cet élément a des marges internes de 1 em. Grâce à box-sizing, aucun débordement n'est généré. On peut cependant noter que la hauteur est impactée.

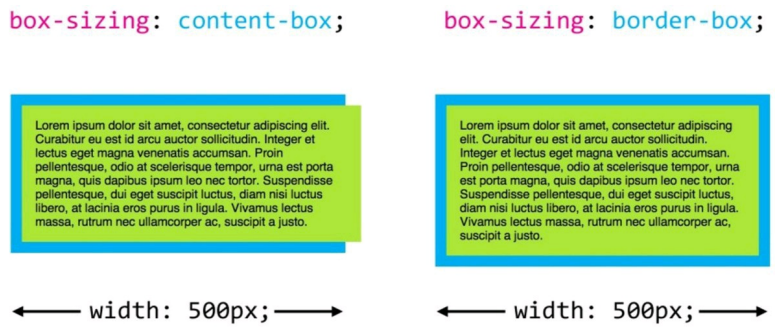


Fig. 5.5 Application de box-sizing

box-sizing	8.0	2.0 -moz-	4.0 -webkit-	10.0	3.1 -webkit-	5.1	9.5	3.2 -webkit-	5.0	10.0	2.1 -webkit-	4.0

Fig. 5.6 Tableau de compatibilité de box-sizing.
box-sizing est assez bien supportée dans les navigateurs,
seules les versions d'IE 7 et inférieures souffrent du manque de compatibilité.

5.2.3. La propriété display

Depuis la première version de CSS, la propriété display permet de modifier le modèle de boîte d'un élément HTML. Pour un besoin spécifique, il est possible de forcer l'affichage d'un élément *inline* en type *block* (display:block;) ou, à l'inverse, de forcer un affichage en ligne (display: inline;). Cela est nécessaire pour dimensionner et styler un lien HTML par exemple.

Ce qui nous intéresse ici, ce sont les autres valeurs de display beaucoup moins connues, comme inline-

block, list-item ou encore table, table-cell ou table-row. Ces valeurs, présentes dans la recommandation CSS2.1 et maintenant bien prises en charge par les navigateurs, simulent d'autres modèles de boîtes et facilitent la gestion de la mise en page. Nous allons nous intéresser à inline-block, puis à table et ses déclinaisons.

inline-block

Cette valeur de la propriété display permet de bénéficier des avantages des deux modèles de boîtes : *block* et *inline*. Un élément peut donc être dimensionné tel un *block*, mais reste en ligne, c'est-à-dire que les éléments restent côte à côte. Cela permet entre autre de pouvoir fixer la taille de plusieurs éléments qui sont voués à s'afficher horizontalement, sans l'utilisation de la propriété float. De plus, un alignement vertical des éléments eux-mêmes devient possible, sans toutefois avoir la possibilité d'un alignement vertical au sein de chaque boîte (figure 5.7).

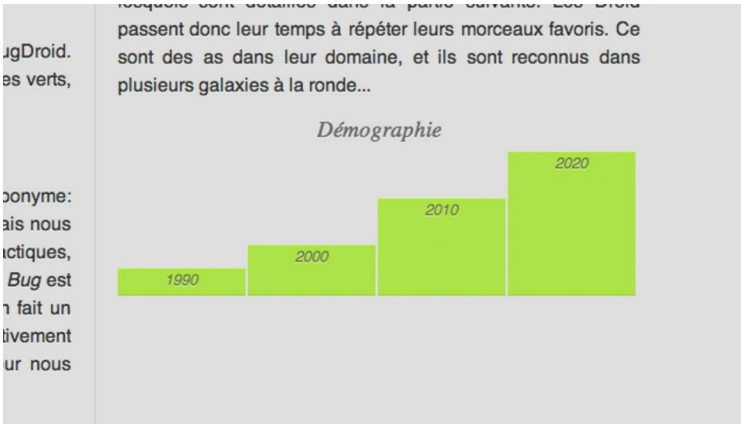


Fig. 5.7 Sur le site de démonstration, inline-block est utilisé pour la création du graphique de la démographie

Prenons le code HTML suivant : une simple liste.

HTML

```
<ul id="chart">
	<li>1990</li><li>2000</li><li>2010</li><li>2020</li>
</ul>
```

L'écriture sur une ligne unique à une importance. Retrouvez l'explication en fin de partie.

Il devient ainsi très facile de créer nos barres en appliquant le display: inline-block sur nos , tout en les dimensionnant (figure 5.8) :

CSS

```
ul#chart li{
	display: inline-block;
	width: 25%;
	height: 20px;
	background: hsl(80,68%,60%);
}
```

Fig. 5.8 Application de display : inline-block

L'utilisation de la règle float, très utilisée dans ce genre de situation, devient alors obsolète. La gestion du flux de la page s'en trouve facilitée. En effet, il est inutile d'utiliser un clear: left sur l'élément suivant pour annuler l'effet du float.

Pour parfaire l'effet, ajoutons une hauteur différente pour chaque élément, et alignons l'ensemble par le bas. Quelques éléments graphiques, comme les styles des textes, la couleur sont également ajoutés.

```
ul#chart li{
        ...
                vertical-align: bottom;
        text-align:center;
        font: italic .8em sans-serif;
        color: rgba(0,0,0,.5);
}
ul#chart li:nth-child(2){
        height: 50px;
}
ul#chart li:nth-child(3){
        height: 70px;
}
ul#chart li:nth-child(4){
        height: 120px;
}
```

Nous ajoutons aussi un espace entre chaque élément de liste, à l'aide des marges externes, tout en diminuant leur largeur de 25 % à 24,5 % (figure 5.9) :

```
ul#chart li{
        ...
        width: 24.5%;
        margin:0 .5% 0 0;
}
```

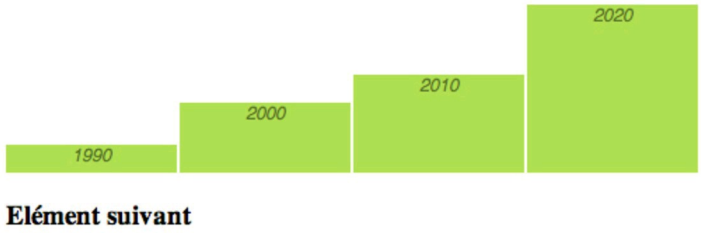


Fig. 5.9 Nos éléments sont positionnés

display: inline-block	8.0	3.0	4.0	3.1	9.0	3.2	10.0	2.1

Fig. 5.10 Tableau de compatibilité de display : inline-block

Cette propriété n'est pas reconnue dans les versions d'Internet Explorer 7 et inférieures. Néanmoins, en ajoutant un display: inline et en donnant le layout[1] à l'élément avec zoom: 1, il est possible de simuler l'effet. Utiliser pour cela les règles conditionnelles.

vertical-align

Pour votre information, voici le rendu de différentes applications de vertical-align sur des éléments affichés en inline-block (figure 5.11, un padding-top a été ajouté sur le troisième élément pour mieux apprécier les différences)

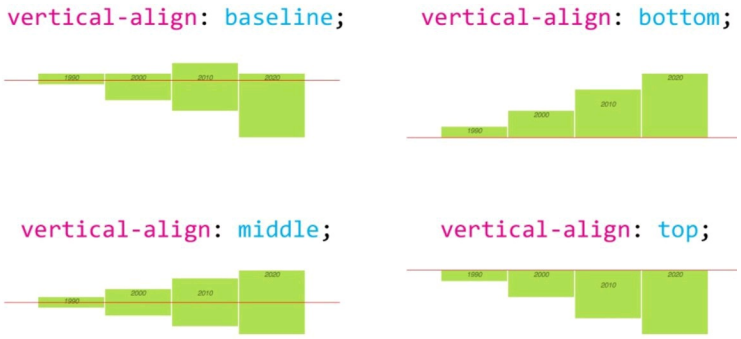


Fig. 5.11 Différents alignements avec vertical-align et inline-block

Nous pouvons noter que vertical-align n'aligne pas le contenu de l'élément, mais bien les éléments entre eux, comme dans le modèle de boîte *inline*. Le modèle de boîte tabulaire, que nous verrons plus loin, permet un centrage vertical du contenu.

white-space

L'utilisation de la propriété inline-block peut générer un décalage d'environ 4 px entre les différents éléments. Ce décalage est provoqué par les espaces présents dans le code source entre deux balises, qui génère un nœud DOM, c'est le problème de *white-space*.

Il existe plusieurs techniques CSS pour remédier à ce problème. Nous pouvons citer en vrac, l'utilisation de marges négatives, la modification de l'interlettrage ou la gestion des espaces avec la propriété white-space. Malheureusement, aucune solution ne semble parfaite dans tous les cas. C'est pourquoi la propriété text-space-collapse pourrait corriger ce problème, mais n'est pour le moment supportée dans aucun navigateur. Nous évoquons cette propriété dans le chapitre sur la typographie.

En attendant, les méthodes fiables passent par la modification du code source, en ne laissant pas de caractères blancs entre deux éléments. Ainsi, il est possible d'écrire les li sur une seule ligne, ou encore ajouter des commentaires HTML entre deux éléments de liste. Sur le site de démonstration, nous avons opté pour l'écriture sur une ligne unique.

Table et ses déclinaisons

Continuons avec les valeurs table et ses dérivés, table-row, table-cell... Ces valeurs de la propriété display nous permettent de simuler une structure de mise en pages en tableaux. En lieu et place de l'utilisation des balises table, tr ou td, il est possible de modifier le comportement d'un élément HTML quelconque pour qu'il se comporte tel un tableau, telle une ligne ou telle une cellule.

La valeur table nous permet donc de simuler un tableau (balise table), la valeur table-row simule une ligne de tableau (balise tr), la valeur table-cell simule une cellule (balise td).

Il devient ainsi possible de profiter des avantages de cette structure sans ses défauts. Par exemple, il

est très facile de positionner des éléments côte à côte et de gérer une hauteur commune. Mais il est encore possible d'aller bien plus loin.

Regardons de plus près le site de démonstration, à la partie *Droid Music*. Cette partie est composée de quatre blocs, côte à côte, dans lesquels une image et un contenu textuel sont présents (figure 5.12).

Fig. 5.12 Partie « Droid Music » du site support

Le code HTML de cette partie est composé d'une liste non ordonnée :

HTML

```
<ul>
	<li>...</li>
	<li>...</li>
	<li>...</li>
	<li>...</li>
</ul>
```

Où chaque élément de liste contient une balise figure, qui elle-même est composée d'une image et d'une figcaption avec le texte :

```
<li>
	<figure>
		<img src="image.png" alt="Texte" />
		<figcaption>
			<h3>Titre</h3>
			<p>Texte</p>
		</figcaption>
	</figure>
</li>
```

Grâce au modèle de boîte tabulaire, nous allons positionner nos éléments sur une ligne unique, et chaque élément occupera automatiquement 25 % de la taille de son parent, car le modèle fonctionne ainsi. Nos éléments li correspondent donc à nos cellules et l'élément ul correspond au tableau (figure 5.13).

CSS

```
ul{
	display: table;
	width: 980px;
	table-layout: fixed;
```

```
}
ul li{
	display: table-cell;
}
```

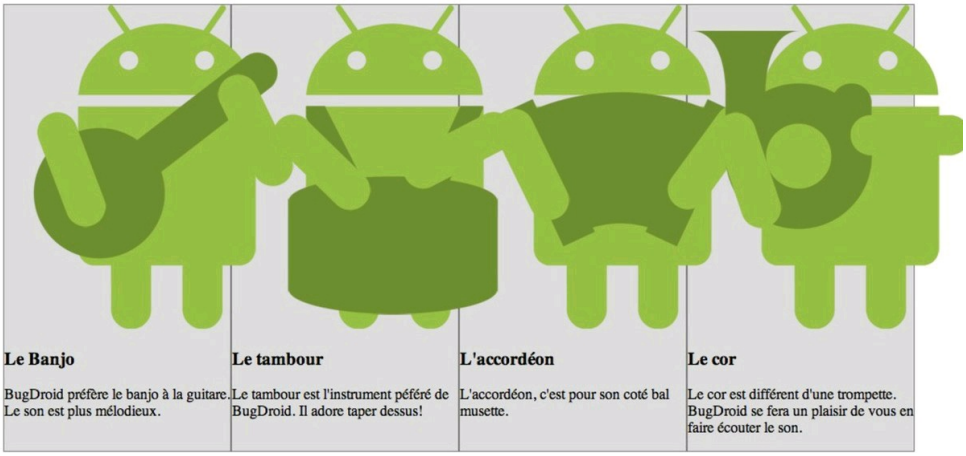


Fig. 5.13 Rendu de display: table
Une couleur de fond ainsi qu'une bordure ont été ajoutés

Nous remarquons qu'il n'est pas nécessaire d'utiliser un display:table-row. Cependant, bien qu'il soit possible d'utiliser un display: table-cell seul, nous vous conseillons l'utilisation du display: table et du table-layout: fixed sur le parent, qui permet de forcer le rendu du tableau à occuper un espace fixe. Avec le table-layout: fixed, les cellules s'adaptent au tableau. Sans, les cellules s'adaptent à leur contenu, et le tableau s'adapte aux cellules.

Continuons notre exemple en fixant la taille de nos images pour qu'aucun débordement ne soit créé. Nous avons choisi une valeur maximum de 80 % (figure 5.14). Ajoutons également un text-align:center sur les li pour parfaire encore le rendu.

```
ul li{
	...
	text-align: center;
}
ul li img{
	max-width: 80%;
}
```

Nous alignons également le contenu texte de figcaption à gauche :

```
ul li figcaption{
	text-align: left;
}
```

Fig. 5.14 Les images mesurent au maximum 80 % de leur parent

Terminons en appliquant un effet de hover sur l'élément survolé. Celui-ci doit s'agrandir en largeur, et les autres éléments doivent se condenser pour ne pas dépasser de leur parent. Là encore, le modèle de boîte tabulaire est parfait puisque l'effet est « automatique ». Nous spécifions donc une largeur de 30 % pour l'élément survolé. Les autres éléments adapteront leur taille (figure 5.15).

```
ul li:hover{
	width: 30%;
}
```

De plus, nous alignons le contenu de chaque cellule en haut de la boîte, avec la propriété vertical-align :

```
ul li{
	...
	vertical-align: top;
}
```

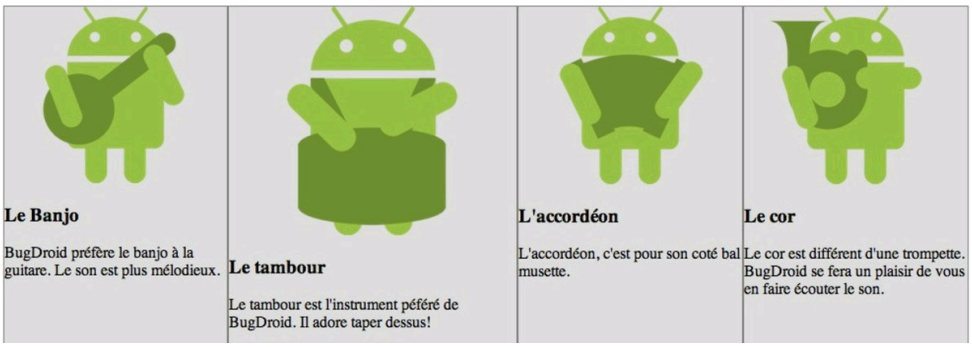


Fig. 5.15 Un élément est survolé

> **Note** : la démonstration présente sur le site utilise un effet de transition sur la taille. Celui-ci est réalisé en CSS, mais implique certaines contraintes, comme l'impossibilité d'animer une valeur qui n'est pas définie au préalable. Rendez-vous au chapitre 7 Transitions et animations pour plus d'infos.

Pour votre information, voici le rendu de différentes applications de vertical-align sur des éléments affichés en table-cell (un padding-top a été ajouté sur le troisième élément pour mieux apprécier les différences). Vertical-align aligne le contenu au sein de celle-ci (figure 5.16).

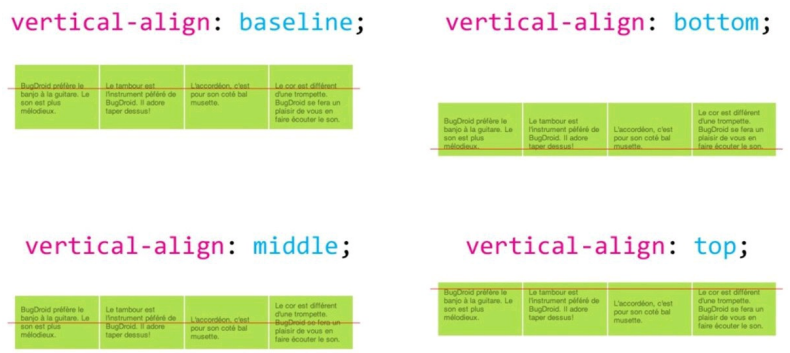


Fig. 5.16 Différents alignements avec vertical-align et table-cell

Nous pouvons conclure cette démonstration en affirmant que les avantages de cette technique sont multiples :

- La séparation du fond (HTML) et de la forme (CSS) est respectée.
- Il n'y a aucun élément qui sort du flux, comme ça serait le cas avec l'utilisation de la propriété float. Il n'y a donc pas besoin d'utiliser clear sur le bloc suivant.
- Les blocs li ont la même hauteur. Si l'un devient plus grand en hauteur (contenu texte plus important par exemple), les autres blocs « frères » s'agrandiront également.
- L'utilisation de bordures (border) ou de marges internes (padding) ne modifie pas la taille des blocs, comme c'est le cas normalement. La taille totale est calculée avec ces valeurs incluses, à l'instar de box-sizing: border-box;.
- Il devient possible de centrer verticalement le contenu du bloc avec vertical-align.

Attention toutefois, les marges externes (margin) ne sont pas applicables sur les éléments affichés en table-cell et en table-row, comme c'est le cas avec les balises td et tr. De plus, les marges internes (padding) ne sont pas applicables sur des éléments en table-row.

C'est pourquoi, l'utilisation des propriétés CSS2.1 spécifiques aux tableaux est obligatoire. Ainsi, la propriété border-spacing doit être ajoutée sur le parent (table) pour décaler les cellules les unes des autres. Dans notre exemple, cela donne :

Décaler les « cellules »

```
ul{
	border-spacing: 5px;
}
```

Il faut pour cela que la propriété border-collapse soit définie à separate, ce qui est le cas par défaut. L'autre valeur possible est collapse.

Aller plus loin !

Nous pourrions nous arrêter là. Mais c'était sans compter sur les autres valeurs possibles. En effet, la spécification ne définit pas moins de dix valeurs pour ce modèle de boîte : table, inline-table, table-row-group, table-header-group, table-footer-group, table-row, table-column-group, table-column, table-cell, table-caption.

Nous avons déjà étudié les trois valeurs « classiques », mais d'autres peuvent se révéler intéressantes.

La valeur inline-table par exemple, permet de créer un tableau qui suit le modèle de boîte *inline*, alors que la valeur table suit le modèle de boîte *block*. Les éléments inclus suivent toujours le modèle de

boîte tabulaire.

Les valeurs table-header-group et table-footer-group sont plus qu'intéressantes. Elles permettent de créer un ensemble de lignes de tableau (tel que table-row-group) qui seront positionnées soit avant (header), soit après (footer) toutes les autres lignes du tableau. Il est donc possible d'inverser l'ordre d'affichage des éléments par rapport au code source, et cela peut être très utile dans certains cas de figure (figure 5.17).

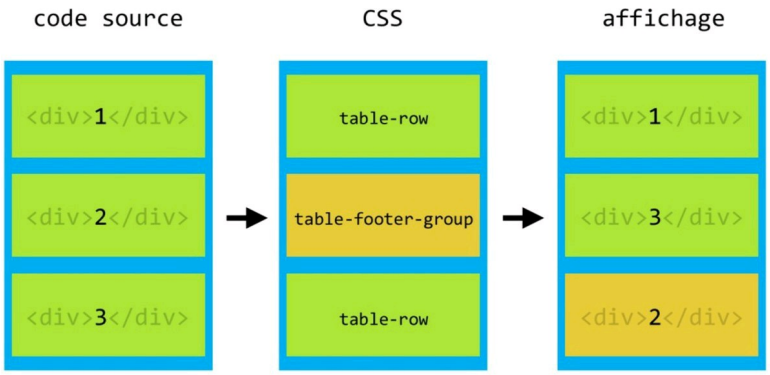


Fig. 5.17 Exemple d'utilisation sur trois blocs

Terminons par la valeur table-caption qui permet d'afficher un élément HTML comme étant la légende du tableau. La propriété caption-side avec les valeurs top ou bottom nous aide à positionner cette légende avant ou après le tableau.

display: table	8.0	2.0	4.0	3.1	9.0	3.2	10.0	2.1

Fig. 5.18 Tableau de compatibilité de display : table

5.3. Positionnements avancés

Bien que CSS2.1 définisse des structures de positionnement intéressantes, la spécification CSS3 introduit de nouveaux systèmes de mise en pages. Ces nouveaux modules tentent de répondre aux évolutions du web qui devient une plateforme applicative : l'ère des pages web s'estompe. De nouvelles politiques de placement doivent alors êtres étudiées afin de se rapprocher de la logique de construction d'une application native sur un système d'exploitation.

Pour cela, CSS3 définit deux nouveaux modules dont nous allons parler à titre expérimental : le **Flexible Box Layout** et le **Grid Module Layout**. En réalité, ce sont plusieurs modules qui sont à l'étude, mais ces deux-là ciblent parfaitement les besoins actuels et futurs et seront très certainement une recommandation à suivre dans quelque temps. Attention tout de même, peu de navigateurs supportent ces modèles de mise en page, et les spécifications sont toujours à l'état de *Working Draft*, voire d'*Editor's Draft*.

Avant de commencer avec ces modules, intéressons-nous à un modèle qui se rapproche plus du média papier, le modèle multi-colonnes ou **Multi-column Layout**, dont la spécification est en *Candidate Recommandation*.

5.3.1. Multi-column layout

Le *Multi-column Layout* est un système de positionnement qui, comme son nom l'indique, permet de répartir un contenu sur différentes colonnes au sein du même bloc, à la façon d'un article de journal (figure 5.19).

Fig. 5.19 Effet article de journal

Les colonnes

Découvrons en détail l'utilisation de cette structure.

Premièrement, nous devons définir le nombre et/ou la largeur souhaitée pour nos colonnes dans notre élément. Pour cela, nous disposons de trois propriétés :

- column-width détermine la largeur optimale de nos colonnes.
- column-count fixe le nombre de colonnes à créer.
- columns : c'est la propriété raccourcie des deux premières.

Attention, column-width est la largeur optimale des colonnes ce qui signifie que le navigateur ne suit pas forcément cette règle. Par exemple, si la valeur de column-width n'est pas un multiple de la taille de l'élément, la valeur choisie est le multiple inférieur. Dans les faits, un élément de 300 px de large avec la règle column-width:120px crée un élément de deux colonnes de 150 px, au lieu de deux colonnes de 120 px et d'un espace vide de 60 px. La règle column-count: 2 produit donc le même résultat ! La règle column-count: 3 crée, elle, trois colonnes de 100 px (ces calculs théoriques ne fonctionnent que si l'espacement entre deux colonnes est fixé à 0) :

Création de deux colonnes (figure 5.20)

```
div{
	width: 300px;
	column-width: 120px;
}
```

est identique à :

```
div{
	width: 300px;
	column-count: 2;
}
```

Création de trois colonnes (figure 5.20)

```
div{
	width: 300px;
```

```
        column-count: 3;
}
```

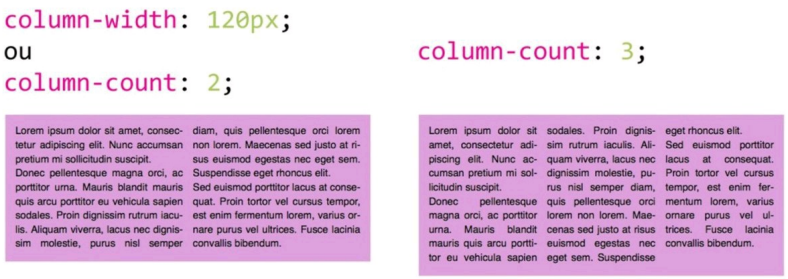


Fig. 5.20 Application du modèle multi-colonnes
Une marge est présente entre les colonnes, pour une meilleure lisibilité

La propriété columns permet d'écrire l'une ou l'autre des propriétés. Ainsi, columns: 120px ou columns: 2 sont tout à fait valides. Attention, si les deux valeurs sont définies, column-count sera considéré comme le nombre maximal de colonnes à créer. Ainsi, columns: 100px 2 crée deux colonnes, bien que 100 px permettrait d'en créer trois.

L'espacement

Il est ensuite possible de gérer l'espace entre les colonnes et d'ajouter une séparation visuelle entre celles-ci. Les propriétés sont les suivantes :

- column-gap : spécifie la taille de l'espace entre deux colonnes. Cet espace s'ajoute de part et d'autre de la limite de colonne. Ainsi, deux colonnes de 145 px de large avec un espace de 10 px occupent une largeur de 145 + 10 + 145 soit 300 px.
- column-rule : ajoute une séparation visuelle entre les colonnes. L'utilisation est la même que celle des bordures: il faut donc absolument définir une couleur, un style et une taille (séparés par des espaces). C'est la notation raccourcie des propriétés column-rule-color, column-rule-style et column-rule-width. Cette séparation se place au centre de la marge définie avec column-gap et n'en augmente pas la taille.

Voici donc un trait pointillé blanc entre deux colonnes séparées de 20 px (figure 5.21) :

Séparation entre les colonnes

```
div{
        width: 300px;
        columns: 2;
        column-gap: 20px;
        column-rule: red dotted 2px;
}
```

Fig. 5.21 Séparation visuelle entre deux colonnes

Contrôle

Les propriétés précédentes suffisent en général pour gérer vos contenus en multi-colonnes. Mais voici d'autres règles éventuellement utiles :

- column-span : permet d'étendre un élément sur plusieurs colonnes. Les valeurs possibles sont none o u all. Lorsqu'un élément est étendu sur plusieurs colonnes, le contenu qui précède est automatiquement réparti entre les colonnes avant l'affichage de cet élément.
- column-fill : gère la méthode de remplissage des colonnes. Les valeurs sont balance ou auto. Soit le contenu est réparti équitablement entre les colonnes, soit non. Cette propriété ne fonctionne qu'avec un média paginé (comme pour l'impression d'un document)
- column-break-before, column-break-after, column-break-inside : ajoute un saut de colonne avant, après ou au sein de l'élément. Permet par exemple de positionner un titre toujours en haut d'une colonne.

Démo

Sur le site de présentation, nous utilisons les multi-colonnes pour la partie *Galaxie Droomble* (figure 5.22). Cette partie est composée d'un texte, d'une image et de sous-titres.

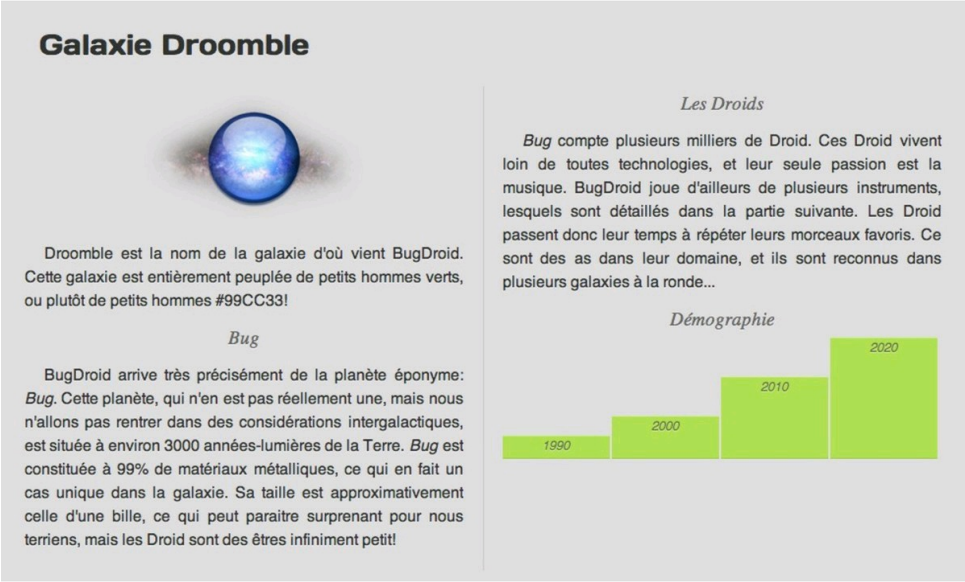


Fig. 5.22 Texte affichés sur plusieurs colonnes

L'affichage de données en multicolonnes pose de réels soucis d'accessibilité sur écran. Par exemple, si les colonnes sont plus hautes que la hauteur visible par l'utilisateur, celui-ci, arrivé en bas d'une colonne, est obligé de remonter en haut pour continuer sa lecture. Notre conseil est de ne pas utiliser les colonnes sur de trop grandes hauteurs.

Multi-column layout	10.0	2.0 -moz-	4.0 -webkit-	3.1 -webkit-	11.1	3.2 -webkit-	11.1	2.1 -webkit-

Fig. 5.23 Tableau de compatibilité de Multi-column layout

Seuls IE et Opéra reconnaissent entièrement ce modèle de positionnement, avec notamment le support de column-break-before, column-break-after et column-break-inside.

5.3.2. Flexible Box Layout

Le *Flexible Box Layout*, ou *Flexbox Layout*, est un système de mise en pages qui permet une gestion sensiblement identique au *block layout*, où chaque élément est positionné par rapport à ses éléments frères. Cependant, son système de positionnement est nettement plus puissant:

- l'ordre d'affichage des éléments peut être modifié dynamiquement, aussi bien horizontalement que verticalement ;
- la taille de chaque bloc peut s'ajuster automatiquement dans l'espace restant, en largeur mais également en hauteur ;
- il est possible d'aligner des blocs horizontalement ou verticalement, sur la largeur ou sur la hauteur de son parent.

Pour faire court, ce mode de positionnement permet une mise en page fluide.

Attention, la spécification de Flexbox Layout a été entièrement modifiée en mars 2011. De plus, étant toujours à l'état WD (Working Draft), il y a de grandes chances pour qu'elle soit encore revue. Les indications qui suivent sont donc à prendre à titre d'information. Néanmoins, le principe de mise en pages reste le même entre les différentes actualisations.

Pour effectuer vos tests, sachez que seul Chrome 17 implémente cette syntaxe, mais certains implémentent l'ancienne. Reportez-vous au tableau récapitulatif pour plus de détails.

display: flexbox

Prenons le code HTML suivant :

HTML

```
<div id="flex">
	<div>1</div>
	<div>2</div>
	<div>3</div>
</div>
```

Déclarons l'élément div#flex comme étant un élément *flexbox* en ajoutant la règle dans le CSS :

CSS

```
#flex{
	display: flexbox;
}
```

Ainsi, l'élément utilisera le modèle de boîte *flexbox*. Les éléments enfants pourront donc être positionnés à l'aide des nouvelles propriétés définies dans cette spécification.

Cette règle est l'équivalent de display: box; dans l'ancienne spécification.

flex-flow

La propriété flex-flow définit la méthode d'affichage des éléments au sein de l'élément *flexbox*. Elle détermine l'axe principal du *flexbox*. Les valeurs possibles sont row, column, row-reverse et column-reverse. Cela permet de positionner les éléments horizontalement en lignes (row), ou verticalement en colonnes

(column), tout en choisissant le sens d'affichage (-reverse). L'ajout éventuel de wrap, wrap-reverse définit le *flexbox* comme étant multi-lignes et permet de choisir le sens d'affichage de ces nouvelles lignes.

La propriété flex-flow est en fin de compte la propriété raccourcie de flex-direction (row, column, row-reverse, column-reverse) et de flex-wrap (nowrap, wrap, wrap-reverse).

Voici les différents résultats de rendus, avec le CSS associé (figure 5.24) :

Affiche les éléments verticalement du haut vers le bas

```
#flex{
	flex-flow: column;
}
```

Affiche les éléments horizontalement de droite à gauche

```
#flex{
	flex-flow: row-reverse;
}
```

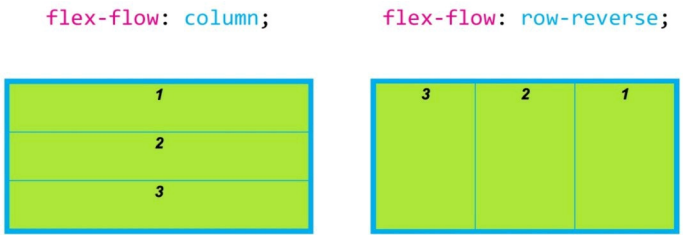


Fig. 5.24 Application de flex-flow

Cette propriété est la combinaison de box-orient et de box-direction dans l'ancienne spécification.

flex-order

La propriété flex-order détermine l'ordre d'affichage des éléments par rapport à l'axe principal, sans se soucier de la position dans le code source. La valeur par défaut est 1. Cette propriété va donc plus loin que flex-flow puisqu'elle permet de choisir la position précise de chaque élément.

Affiche la deuxième div en première position (figure 5.25)

```
#flex{
	flex-flow: column;
}
#flex div:nth-child(1), #flex div:nth-child(3){
	flex-order: 2;
}
```

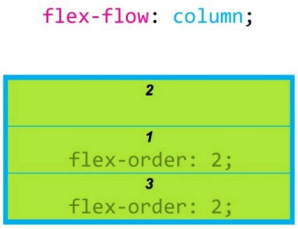


Fig. 5.25 Application de flex-order

Cette propriété est l'équivalent de box-ordinal-group dans l'ancienne spécification.

flex-pack

La propriété flex-pack spécifie l'alignement des éléments sur l'axe principal défini par flex-flow (figure 5.26). Elle s'utilise sur l'élément *flexbox*. Les valeurs possibles sont:

- start : les éléments sont alignés au départ (à gauche, à droite, en haut ou en bas) de la boîte. Une marge est créée à l'opposé.
- end : les éléments sont alignés à la fin (à gauche, à droite, en haut ou en bas) de la boîte. Une marge est créée à l'opposé.
- center : les éléments sont positionnés au centre de la boîte. Une marge de chaque côté est créée.
- justify : les éléments sont alignés de part et d'autre (à gauche et à droite ou en haut et en bas) de la boîte. Des marges de même taille sont créées entre les éléments.
- distribute : des marges identiques sont créées de part et d'autre de chaque élément.

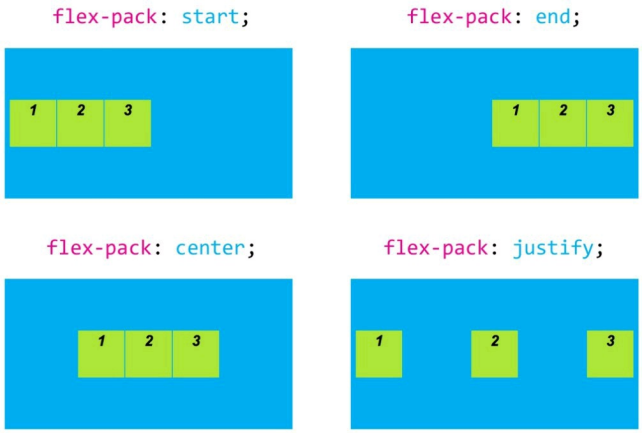


Fig. 5.26 Application de flex-pack sur l'axe principal défini par flex-flow: row.

Attention, si flex-flow: row-reverse est précisé, start aligne les éléments à droite, et end les aligne à gauche. Si flex-flow: column est précisé, start aligne les éléments en haut, et end en bas, et inversement pour flex-flow: column-reverse. Il en est de même pour toutes les propriétés qui suivent.

Cette propriété est l'équivalent de box-pack dans l'ancienne spécification.

flex-align et flex-item-align

La propriété flex-align spécifie l'alignement des éléments au sein du parent *flexbox*, en suivant l'axe secondaire, c'est-à-dire l'axe perpendiculaire à l'axe principal (figure 5.27). Elle s'utilise sur l'élément *flexbox*. Les valeurs possibles sont start, end, center, baseline et stretch. La propriété flex-item-align fonctionne de manière identique mais pour un seul élément. Elle s'utilise donc sur un enfant de *flexbox*.

L'utilisation de ces propriétés est identique à flex-pack, mais pour l'alignement sur l'axe secondaire.

Dans les exemples suivants, des hauteurs différentes ont été spécifiées et un padding-top a été ajouté sur la deuxième div :

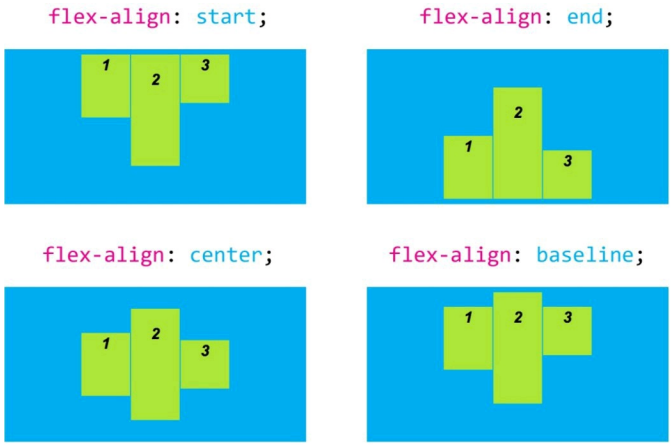


Fig. 5.27 Application de flex-align sur l'axe secondaire
Un padding-top a été ajouté sur le deuxième élément

Cette propriété est la combinaison de box-align et de box-flex dans l'ancienne spécification.

flex-line-pack

La propriété flex-line-pack définit l'alignement vertical des multilignes d'éléments sur l'axe secondaire. Elle s'utilise sur l'élément *flexbox*. Les valeurs possibles sont start, end, center, justify, distribute ou stretch.

Si le mot-clé wrap (ou wrap-reverse) est défini sur la propriété flex-flow, cela signifie que le *flexbox* est multiligne. Ainsi, les éléments ne débordent plus de leur parent mais se positionnent sur une nouvelle ligne de l'axe principal. flex-line-pack modifie donc l'alignement de ces multiples lignes.

L'utilisation est identique à flex-pack.

flex()

La fonction flex() permet de spécifier la méthode de calcul de la taille des éléments enfants de *flexbox*. Pour rester simple, elle permet de fixer la taille des éléments tout en leur demandant de s'adapter au cas où de la place serait disponible. La valeur définie dans la fonction flex() fait office de valeur minimale (figure 5.28).

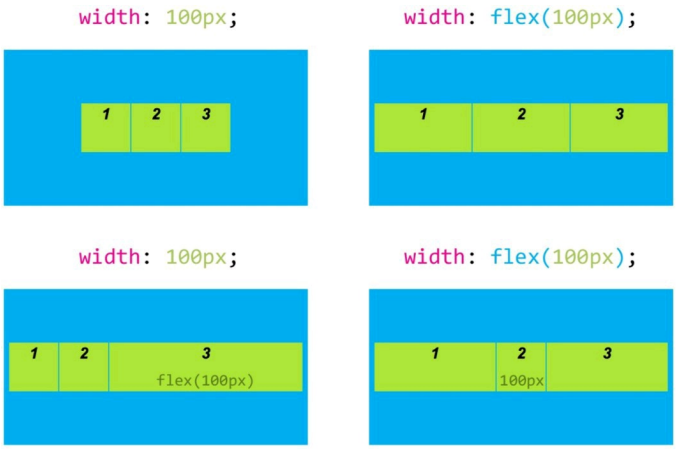


Fig. 5.28 Application de flex()

Lors d'un affichage réduit ou d'un redimensionnement, les boîtes qui utilisent flex() s'adaptent, avec une largeur minimale de 100 px.

Cette fonction est actuellement vouée à disparaître au profit d'une propriété flex. Le principe de fonctionnement reste cependant le même: permettre aux éléments d'avoir une taille flexible.

Flexible box layout	10.0 -ms-	2.0 -moz-	4.0 -webkit-	3.1 -webkit-		3.2 -webkit-		2.1 -webkit-

Fig. 5.29 Tableau de compatibilité de Flexible box layout

Attention, la syntaxe telle que présentée dans cet ouvrage n'est reconnue que dans Chrome 17. La figure 5.29 précise les navigateurs qui supportent l'ancienne syntaxe.

5.3.3. Grid layout

Le *Grid Layout* est un système de mise en pages dont le principe est de découper une page web en lignes et en colonnes (une grille), afin de positionner et de dimensionner les éléments HTML dans cette grille. Cela permet notamment une mise en page fluide qui occupe la totalité de l'écran de l'utilisateur, que celui-ci soit sur un appareil mobile ou sur un ordinateur de bureau. Les éléments sont donc positionnés au sein des cases de cette grille, et ce, sans se préoccuper de leur position dans le code source, ce qui est une réelle évolution par rapport à tous les autres systèmes. Des règles de positionnement permettent ensuite de pouvoir aligner le contenu à l'intérieur d'une case.

L'énorme avantage de ce système de placement réside dans la capacité à positionner un élément à un endroit de la page, tout en le conservant dans le flux. La facilité d'utilisation permet une gestion efficace et propre de l'architecture d'un site (en prenant soin de conserver un ordre logique dans le code source : éléments importants vs éléments secondaires).

Attention, la spécification de Grid Layout est à l'état embryonnaire. Les informations données ici sont à prendre à titre d'informations et d'expérimentations.

display: grid

Définissons en premier lieu le code HTML :

HTML

```
<div id="grid">
	<header>Titre</header>
	<nav>Menu</nav>
	<article>Texte</article>
	<aside>Liens</aside>
</div>
```

Spécifions à présent l'utilisation d'une grille, avec la valeur grid :

CSS

```
#grid{
	display: grid;
	width: 800px;
	height: 500px;
}
```

Le modèle de mise en pages *grid*, avec le display: grid est donc appliqué. La taille de notre élément est

fixée à 800 px sur 500 px.

grid-columns, grid-rows

Créons maintenant la structure de notre grille : elle sera composée de trois lignes et de trois colonnes. Les propriétés grid-columns et grid-rows (avec un « s ») sont prévues pour cela.

```
#grid{
	diplay: grid;
	grid-columns: 200px 1fr 200px;
	grid-rows: 100px 1fr;
}
```

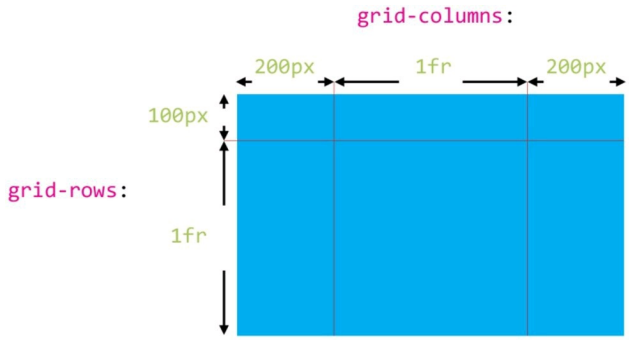


Fig. 5.30 Création de trois lignes et de trois colonnes

Dans ce cas, les colonnes de gauche et de droite mesurent 200 px tandis que la colonne centrale adapte sa taille à la place restante (1 fr signifiant une fraction de l'espace restant, figure 5.30). Pour les lignes, la première mesure 100 px de hauteur, la deuxième l'espace restant du parent. Ces règles s'appliquent sur l'élément en display: grid.

grid-column, grid-row

Maintenant que notre grille est créée, il nous suffit de positionner nos éléments HTML dans cette grille, à la manière d'un jeu de bataille navale. Les propriétés grid-column et grid-row (sans « s ») définissent l'adresse de la case, composée du numéro de la colonne et du numéro de la ligne.

L'élément header est positionné en 1,1 (en haut à gauche de la grille), l'élément nav est positionné en 1,2 (en dessous de header), l'élément article est positionné en 2,2 (à droite de nav) et l'élément aside est placé en 3,1 (dernière colonne à droite de header) (figure 5.31)

Positionnement des éléments

```
header{
	grid-column: 1;
	grid-row: 1;
}
nav{
	grid-column: 1;
	grid-row: 2;
}
article{
	grid-column: 2;
	grid-row: 2;
}
aside{
```

```
        grid-column: 3;
        grid-row: 1;
}
```

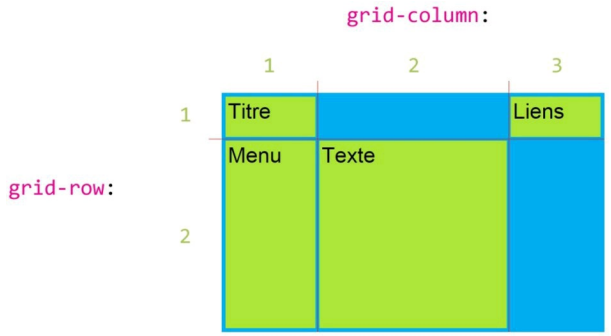


Fig. 5.31 Placement des éléments dans les cases

grid-column-span, grid-row-span

Les propriétés grid-column-span et grid-row-span étendent un élément HTML pour qu'il occupe plusieurs cases, dans le sens horizontal et/ou dans le sens vertical, par rapport à sa position définie avec grid-column et/ou grid-row. Le principe est identique aux attributs colspan et rowspan en HTML (figure 5.32).

L'élément header est étendu sur trois colonnes, et l'élément article est étendu sur deux colonnes. La valeur par défaut est 1 (pour column et row).

```
header{
        ...
        grid-column-span: 3;
}
article{
        ...
        grid-column-span: 2;
}
```

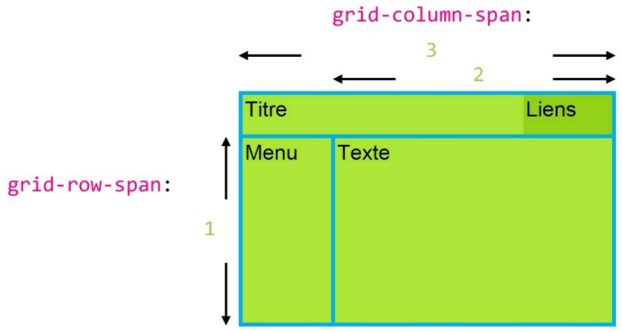


Fig. 5.32 Les éléments sont étendus

Nous remarquons que l'élément header est étendu sur trois colonnes, et par conséquent son contenu est affiché sous l'élément aside. Dans l'exemple, nous avons modifié la couleur d'arrière-plan pour symboliser aside, mais il n'est pas obligatoire. Au contraire, sans background, nous verrions la couleur de fond de l'élément header, ce qui peut s'avérer très utile.

grid-column-align, grid-row-align

Les propriétés grid-column-align et grid-row-align permettent l'alignement au sein d'une case. Les valeurs

possibles pour ces deux propriétés sont start, end, center ou stretch. Il est donc possible de gérer un alignement vertical et horizontal au sein des éléments, ainsi qu'un étirement de la cellule à occuper tout l'espace avec stretch.

La valeur par défaut est stretch, ce qui implique que la cellule est entièrement recouverte par l'élément. Si l'une des autres valeurs est utilisée, la taille de l'élément s'adapte à son contenu.

L'élément header est aligné au centre de la ligne, nav est aligné en fin de colonne et article est aligné centré sur les deux axes (figure 5.33).

```
header{
        ...
        grid-row-align: center;
}
nav{
        ...
        grid-column-align: end;
}
article{
        ...
        grid-column-align: center;
        grid-row-align: center;
}
```

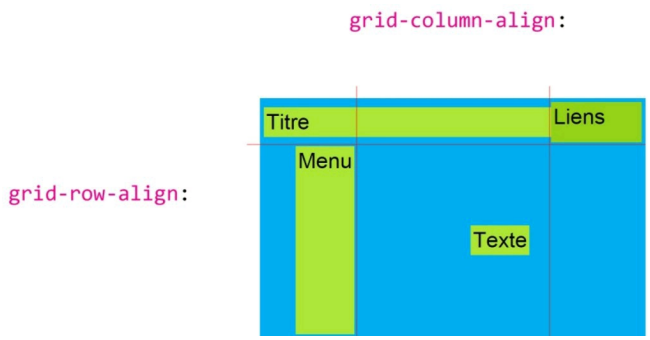


Fig. 5.33 Alignement des éléments dans la grille

grid-layer

La propriété grid-layer modifie l'ordre d'affichage des éléments. Par exemple, si plusieurs éléments HTML sont positionnés dans une même case de la grille, ou si les éléments se superposent, l'ordre d'affichage est celui du code source (l'élément le plus bas dans le code est affiché devant les autres). grid-layer permet de modifier cela. Cette propriété a une utilisation identique à z-index pour les éléments positionnés hors du flux.

template

Le *Grid Layout Module* définit également un système de positionnement à base de *templates*. Ce système reprend les fondamentaux d'un autre module CSS3 : le *Template Layout Module* et ajoute le principe des cellules nommées.

Deux nouveaux principes de placement sont donc ajoutés :

- La grille peut être définie avec des lettres. Par exemple, grid-template: "aa" "bc" "dd"; définit trois lignes de deux colonnes.
- La grille est définie avec des tailles auxquelles sont ajoutés des noms. Il est ainsi possible

d'utiliser cette notation : grid-rows: "entete" 200px "contenu" 1fr;.

L'avantage de ces techniques réside dans le placement des éléments. En effet, il devient possible de les positionner en faisant appel à une cellule nommée, comme par exemple grid-column-row: "entete";. Pour le modèle basé sur les *templates*, la règle grid-cell: "b"; place l'élément dans la première case de la deuxième ligne.

Cette notation permet notamment de simplifier le placement des éléments lors de la création d'un site web qui s'adapte à différentes résolutions. Cependant, ces règles sont à l'état de brouillon et aucun navigateur ne les implémentent. Mais cela augure de bonnes choses pour le futur de la mise en page sur le web.

Grid layout	10.0 -ms-							

Fig. 5.34 Tableau de compatibilité de Grid layout

Le Grid layout n'est supporté actuellement que sur IE10, et en version préfixée. En revanche, l'écriture à base de template n'est pas supportée. De son côté, Webkit travaille à l'implémentation.

Ce qu'il faut retenir...

Nous voici arrivés à la fin de ce chapitre. Nous avons survolé les différentes méthodes de placement des éléments et de gestion de la mise en page. CSS3 nous offre de nouvelles techniques qui paraissent vraiment performantes. Cependant, il paraît évident qu'il est trop tôt pour baser un design exclusivement sur ces techniques car la majorité des navigateurs ne supportent pas ces modèles de positionnement. Mais tout évolue très vite. Les éditeurs ont engagé une mise à jour régulière et automatique de leur navigateur, ce qui permet au web d'évoluer plus rapidement qu'auparavant. Microsoft investit également dans ces nouveaux systèmes, puisque leur nouvel OS Windows 8 en version Métro, est entièrement basé sur ces schémas de positionnement. Nul doute que nous allons assister à l'essor des modèles de mise en page de CSS3 !

En attendant, les modèles définis dans CSS2.1 tel que le modèle tabulaire, nous permettent de créer des mises en pages complexes, qui peuvent s'adapter facilement aux différentes tailles d'écrans, tout en contrôlant le flux de nos pages.

D'ailleurs, nous allons poursuivre cet ouvrage par un chapitre consacré aux périphériques mobiles. Nous allons voir les techniques qui, associées à ces nouveaux schémas, permettent de réaliser des structures qui s'adaptent en fonction de la taille, de la résolution ou même de l'orientation de l'écran.

Notes

[1] http://www.journaldunet.com/developpeur/tutoriel/css/070109-dom-css-ie-haslayout.shtml

Le mobile

Objectifs

La façon de créer un site internet a changé. La façon de naviguer sur internet a changé.

Personne ne pouvait prévoir l'arrivée massive des téléphones mobiles, smartphones et autres tablettes, connectés eux aussi à internet. Personne ne peut également prévoir de quoi l'avenir du web sera fait. L'objectif de ce chapitre est de vous présenter les nouvelles stratégies en matière d'élaboration d'un site web et en quoi CSS peut nous aider.

6.1. Pensez différemment

Aujourd'hui, plus que jamais, il faut qu'un site puisse être accessible depuis des appareils de plus en plus différents. Les écrans d'ordinateurs ont une taille d'écran qui grandit encore alors que, d'un autre côté, l'internet sur mobile progresse à une allure folle. Et cela sans compter sur nos écrans de télévision, nos voitures, nos réfrigérateurs et pourquoi pas nos montres bientôt tous connectés au web (figure 6.1). Nous ne pouvons plus nous contenter de fournir à nos visiteurs un site au design unique, fixe, où l'utilisateur est obligé de zoomer, de scroller, de pincher (ou d'utiliser un autre anglicisme encore inconnu), afin d'accéder à notre contenu. La manière de naviguer est également différente : tactile, avec une souris, à l'aide d'une manette de jeu ou avec un autre accessoire révolutionnaire, tous les cas de figures sont désormais envisageables.

Fig. 6.1 Le web aujourd'hui. Et demain ?

C'est pourquoi, adapter son design pour une utilisation et un affichage efficaces sur tous ces appareils est devenu indispensable. Mais comment réaliser cette tâche ? La solution vient avec CSS3 et le

Responsive Design[1].

L'idée est simple : plutôt que de développer une version mobile dédiée, avec de nouveaux squelettes HTML qui utilisent des feuilles de styles et des médias alternatifs, nous allons ajuster notre design actuel avec du CSS, pour que celui-ci s'adapte à différentes plateformes. Le **Responsive design**, c'est une façon de réfléchir, plutôt qu'une technique de design à part entière. Cette réflexion porte aussi bien sur le mobile que sur toute autre plateforme disponible actuellement ou future. Cependant afin de poser les bases de cette bonne pratique nous allons considérer ici le cas du mobile comme une référence.

CSS3 nous propose des outils pour pouvoir mettre en place cette nouvelle approche. Afin de créer une cohérence entre nos différentes versions de site, voyons d'abord comment adapter son design à un site mobile, puis à l'aide des *media-queries*, examinons les différentes possibilités qui s'offrent à nous pour gérer des affichages variés.

6.2. Adaptez pour le mobile

Vous vous êtes certainement posé la question de savoir comment les utilisateurs nomades voyaient votre site et comment ils naviguaient dessus (figure 6.2). Sans rentrer dans le détail des innombrables marques et modèles de téléphones portables ou tablettes, ni dans les multiples implémentations de leurs navigateurs, nous pouvons classer la méthode de rendu d'une page web de deux façons :

- Le contenu est affiché en taille réelle, ce qui signifie qu'un pixel écran correspond à un pixel CSS. L'utilisateur est alors obligé de se déplacer au sein de la page pour accéder au contenu. La page globale n'est pas visualisée dans son ensemble.
- La largeur du contenu est calculée pour être ajustée à la largeur de l'écran. Un pixel écran correspond alors à plusieurs pixels CSS. La page est visualisée dans son intégralité mais l'utilisateur est souvent obligé de zoomer pour accéder à l'information. Les dernières générations de mobiles utilisent cette méthode de rendu.

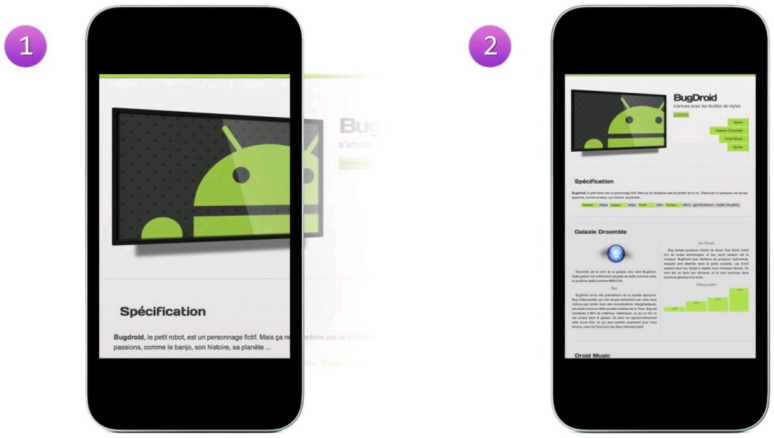


Fig. 6.2 Différence de rendu d'une page web

La gestion du comportement du mobile vis-à-vis du rendu de la page passe par l'utilisation d'une balise meta, la meta viewport. Mais qu'est-ce que le *viewport* ?

6.2.1. Définition du viewport

Le *viewport* est la zone fictive utilisée par les appareils mobiles pour l'affichage d'une page web. La

plupart du temps (mais cela dépend de la plateforme et du navigateur), le viewport est dynamique. Cela signifie qu'il peut s'agrandir si la page à afficher est plus large que sa dimension initiale, afin de faire tenir cette page dans une taille d'écran réduite. Prenons l'exemple de l'iPhone première génération : la taille physique de l'écran est de 320 px de large alors que celle de son *viewport* est de 980 px. Les pages peuvent alors être affichées à l'intérieur de cette zone pour être visualisée dans leur intégralité. Le viewport peut donc être différent du screen-width, qui correspond à la largeur réelle de l'écran, et du device-width, qui est la largeur déclarée de l'écran (figure 6.3). Pour l'iPhone 4 par exemple (écran rétina, haute résolution), le screen-width est de 640 px alors que le device-width est de 320 px, ce qui signifie que 1 px CSS correspond à plusieurs pixels écrans. Le viewport est quant à lui toujours fixé à 980 px par défaut sur un iPhone 4.

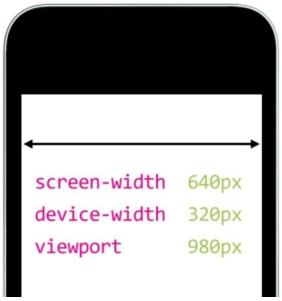


Fig. 6.3 Un écran, trois largeurs

6.2.2. Modifiez le viewport

Comme vu précédemment, le viewport peut prendre des valeurs différentes, sans pour autant que cela soit bénéfique pour nous. En effet, ce n'est pas dans notre intérêt que le mobile dont le viewport est configuré à 980 px de large offre un aperçu de notre site totalement illisible, car l'utilisateur sera au final obligé de zoomer pour atteindre l'information. Mais comment faire pour offrir une version lisible, adaptée à la taille de l'écran sans pour autant refaire une feuille de style dédiée à un appareil en particulier ? La première partie de cette réponse passe par le paramétrage du viewport à l'aide d'une balise meta. Cette balise, à placer dans le head de votre document permet de spécifier une valeur pour le viewport, en remplacement de celle par défaut. En voici la syntaxe que nous détaillerons ensuite :

```
<meta name="viewport" content="width=320,initial-scale=1" />
```

La syntaxe est classique pour une balise meta. Attention toutefois, le code de l'attribut content n'est pas du CSS, par conséquent les différentes valeurs sont séparées par des virgules, en lieu et place de points-virgules. Les deux-points sont remplacés par le signe égal.

Voyons à présent les options disponibles pour gérer ce *viewport*. Commençons par la valeur width qui permet de gérer la largeur de celui-ci. Fixons cette valeur comme ceci :

```
<meta name="viewport" content="width=320" />
```

Avec cette syntaxe, le viewport est fixe pour toutes les résolutions d'écrans. Par exemple, si nous fixons la largeur à 320 px et que le screen-width est de 320 px, l'affichage est cohérent. Mais si nous pivotons l'écran le viewport reste à 320 px alors que le screen-width a sûrement augmenté car la plupart des écrans sont rectangulaires. L'effet est le même pour un périphérique dont le screen-width est plus élevé que 320 px. Le résultat est alors un agrandissement des contenus, ce qui n'est pas idéal (figure 6.4).

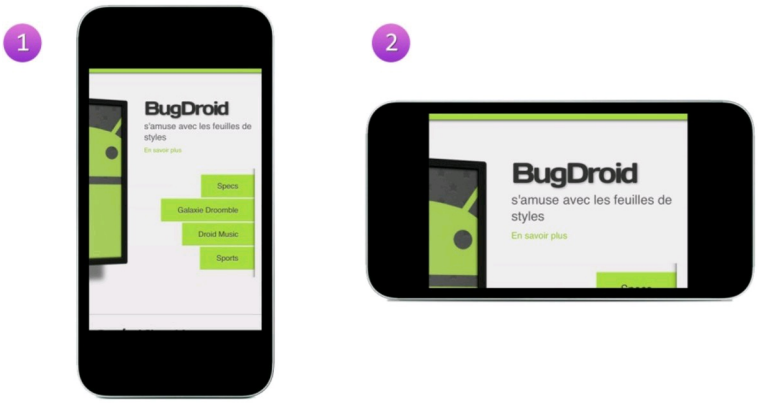


Fig. 6.4 Rendu d'un viewport fixé à 320 px

Pour une gestion plus fine, utilisons alors le mot clé device-width, comme suit :

```
<meta name="viewport" content="width=device-width" />
```

Maintenant, notre viewport est fixé à la largeur déclarée de notre mobile (figure 6.5). Le viewport ne sera donc plus fixe mais calculé en fonction des appareils. Et lors de la rotation de l'écran par l'utilisateur, le viewport est également modifié. Cette syntaxe est idéale dans la majorité des cas.

Fig. 6.5 Rendu d'un viewport fixé à la taille déclarée du téléphone

La hauteur du viewport est également modifiable, mais moins utile. Cependant, dans des cas très spécifiques, tels des jeux ou des applications en ligne, il est nécessaire de le modifier. Pour cela, nous utiliserons height avec une valeur fixe ou avec la constante device-height.

6.2.3. Dépassement du viewport

Que se passe-t-il maintenant si un élément de votre page web est fixé à 600 px alors que votre *viewport* est de 320 px ? La réponse : le viewport sera agrandi automatiquement (le viewport est dynamique), ce qui peut poser quelques problèmes. Pour cela, ajoutons dans notre balise meta la règle initial-scale :

```
<meta name="viewport" content="width=device-width,initial-scale=1" />
```

Cette règle définit le facteur d'agrandissement du viewport. Par défaut, celui-ci est calculé pour que chaque page s'ajuste à la largeur de l'écran, même si cette largeur dépasse la taille du viewport. Dans le cas d'un site mobile, le fait de fixer initial-scale à 1 provoque le débordement des éléments du site dont la taille est plus large que celle du viewport, ainsi le contenu ne subit aucune réduction (figure 6.6).

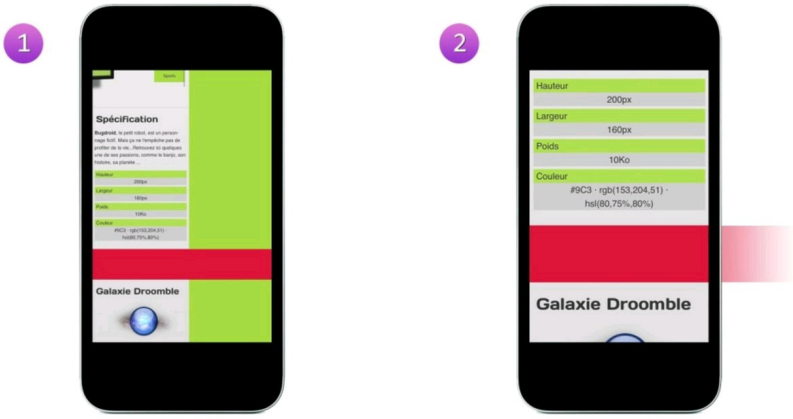


Fig. 6.6 Différence de initial-scale

6.2.4. Autres réglages du viewport

Les règles minimum-scale et permettent de fixer les valeurs de zoom minimum et maximum d'affichage de votre site. Les appareils mobiles ont leurs propres valeurs par défaut. Sur Safari Mobile par exemple, les valeurs par défaut sont respectivement 0,25 et 1,6.

Il est également possible de spécifier la règle user-scalable à yes ou no. Le fait d'ajouter cette règle avec la valeur no interdit à l'utilisateur d'agrandir ou de réduire la page web. L'utilisation d'une telle règle doit se faire pour des besoins très spécifiques, comme par exemple la création d'une application mobile dédiée. Sans cette précaution, de très importants (et évidents) problèmes d'accessibilité sont inévitables. Il existe d'ailleurs une option dans certains navigateurs mobiles pour ignorer cette règle.

Sur la plateforme Android uniquement, une autre fonctionnalité permet de forcer le viewport du périphérique à utiliser une résolution précise, ou à utiliser sa propre résolution, c'est la règle target-densitydpi. L'objectif d'une telle règle est de pouvoir utiliser correctement les media-queries basées sur la résolution afin de cibler les appareils ayant une définition particulière.

6.2.5. Viewport en CSS

Changer la façon dont un site est visualisé sur tel ou tel appareil relève du rendu et non du formatage de la page, cela implique donc l'utilisation de CSS. Cependant pour modifier le viewport, nous devons utiliser une balise HTML, ce qui n'est pas logique. C'est pourquoi, un nouveau module CSS 3 a vu le jour à l'initiative d'Opéra pour pallier ce problème: le *CSS Device Adaptation*.

L'ambition de ce module est simple: reprendre toutes les fonctionnalités de la meta viewport en CSS. Il est donc possible d'utiliser les différentes règles vues plus haut, mais en utilisant la syntaxe CSS, à l'intérieur d'une règle @viewport, comme ceci:

Définir la largeur du viewport à la largeur déclarée de l'écran

```
@viewport{
	width: device-width;
}
```

Ajouter le zoom par défaut (initial-scale à 1)

```
@viewport{
	width: device-width;
	zoom: 1;
```

```
}
```

Les règles initial-scale, minimum-scale, maximum-scale et user-scalable sont pour l'instant renommées respectivement zoom, min-zoom, max-zoom et user-zoom.

@viewport							11.0	

Fig. 6.7 Tableau de compatibilité de @viewport

6.2.6. Unités

De nouvelles unités relatives à la taille du viewport sont disponibles. Ces unités permettent d'aller plus loin que la simple utilisation des pourcentages puisqu'elles sont relatives à la partie visible de la page. Un élément dont la taille est fixée avec ces unités ne subit pas de modification de taille même si la page est agrandie. Ces unités peuvent êtres comparables à position: fixed, mais pour les dimensions.

La spécification définit donc trois unités :

- vw : 1/100ème de la largeur du *viewport ;*
- vh : 1/100ème de la hauteur du *viewport ;*
- vmin : 1/100ème de la plus petite dimension du *viewport.*

Par exemple, il est donc possible d'obtenir un élément qui soit toujours carré et qui occupe le maximum de l'espace du viewport et ce, quelles que soient ses proportions (figure 6.8).

```
div{
        width: 100vmin;
        height: 100vmin;
        background: firebrick;
}
```

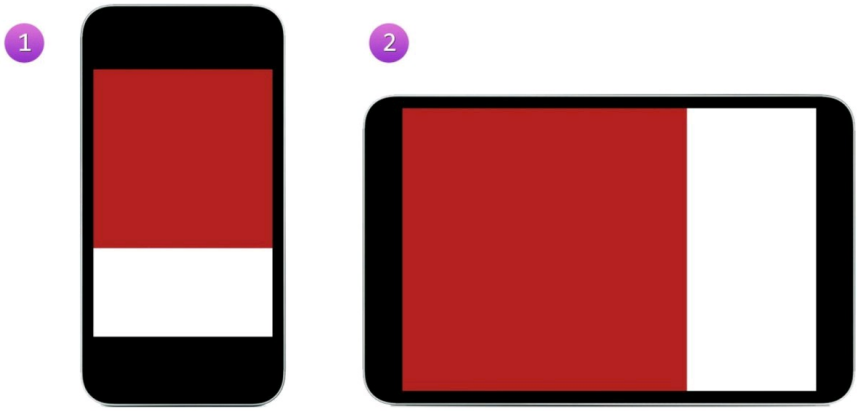


Fig. 6.8 Les dimensions de l'élément sont égales à la plus petite dimension du viewport

vw, vh, vmin	9.0		20.0					

Fig. 6.9 Tableau de compatibilité des unités vw, vh, vmin

6.3. Media-queries

Nous venons de voir qu'il était possible d'adapter la partie visible des différents médias pour qu'ils se comportent de façon homogène. Mais cela ne règle en rien la lisibilité de notre contenu. En effet une mise en page, qu'elle soit dite « fluide » ou bien de taille fixe sur un écran d'ordinateur de taille supérieure à 10" ne sera pas rendue correctement sur un écran qui annonce 320 px de large. Il faut donc adapter le design de la page à la dimension du média. Une des solutions proposées par CSS3 est l'utilisation de styles spécifiques, conditionnés par différents critères propres au périphérique. Ce sont les **media-queries**, désignées par la règle @media.

Ces requêtes sur les médias existent déjà depuis HTML 4.01 et vous les connaissez peut-être sous la forme d'une balise link dans l'en-tête HTML, précisant une feuille de styles spécifique à l'impression de votre page.

```
<link rel="stylesheet" media="print" href="print.css" type="text/css" />
```

Depuis CSS2.1, il est tout à fait possible d'intégrer cette règle au sein d'une feuille de styles avec la syntaxe suivante pour définir un style propre à ce média :

Règles spécifiques pour le média impression

```
@media print{
	html{
		color: midnightblue;
	}
}
```

Le tableau 6.1 présente la liste des valeurs possibles pour l'attribut media.

Tab. 6.1 Liste des valeurs possibles pour l'attribut media

Media	Définition
all	tous les médias
braille	appareil de diffusion tactile en braille
embossed	impression relief en braille
handled	petits écrans d'appareils portables. Attention, la plupart des mobiles actuels ne se considèrent pas comme média handled mais screen !
print	impression par page
projection	projection par page
screen	écran
speech	synthétiseur vocal
tty	écran de type télétexte (minitel, terminaux...)
tv	écran de téléviseur par page avec audio

Associés à ces médias, CSS3 introduit les *media-features*. Ce sont des propriétés qui se rapportent aux capacités du navigateur. La plupart du temps elles sont dérivées en une version préfixée d'un min- ou max- pour tester une plage et non seulement une valeur. L'utilisation des parenthèses pour entourer chaque propriété est obligatoire.

Le tableau 6.2 présente la liste complète telle que décrite dans la *CR Media Queries*.

Tab. 6.2 Liste complète des media-features

Features	Définition
width	la largeur de l'écran (viewport)
height	la hauteur de l'écran (viewport)
device-width	la largeur du périphérique
device-height	la hauteur du périphérique
orientation	périphérique en mode portrait (portrait) ou paysage (landscape)
aspect-ratio	le rapport largeur/hauteur
device-aspect-ratio	le rapport largeur/hauteur du périphérique
color	le nombre de bits utilisés pour définir la couleur
color-index	le nombre de couleurs du périphérique
monochrome	le nombre de bits utilisés (niveau de gris)
resolution	la densité de pixels du périphérique en dpi ou dpcm
Scan (progressive ou interlace)	le balayage de l'écran de la TV
grid	le type d'écran (télétexte ou raster)

Les *media-queries* sont en fait des tests. Le système est interrogé pour chaque valeur de la règle @media et ce dernier renvoi vrai ou faux. Le résultat booléen est alors calculé pour savoir si la règle doit être appliquée. Il est possible d'utiliser des opérateurs booléens afin de tester plusieurs valeurs : une virgule (,) représente un OU logique et le mot clé and représente bien entendu le ET logique.

Nous pouvons donc tester assez finement le périphérique sur lequel la page est visionnée pour lui appliquer des règles spécifiques.

6.3.1. Syntaxe

Voici la syntaxe permettant de définir des styles pour tous les écrans, c'est-à-dire les écrans informatiques OU les écrans de mobiles OU les écrans de télévisions (nous vous rappelons que les smartphones sont considérés comme écrans informatiques et non handheld!)

```
@media screen, handheld, tv{
	body{
		background: #222;
		color: white;
	}
```

```
}
```

Voici un autre exemple introduisant le test d'une propriété. Le style est donc appliqué si et seulement si le média est de type screen et qu'il est en orientation portrait. C'est-à-dire si la largeur est inférieure à la hauteur :

```
@media screen and (orientation:portrait){
	body{
		background: #222;
		color: white;
	}
}
```

Notons une particularité de la syntaxe, la valeur all pour le type de média est optionnelle. Il est donc possible de tester une propriété pour l'ensemble des types de médias simplement comme ceci :

```
@media (color){
	p{
		color: crimson;
	}
}
```

est équivalent à :

```
@media all and (color){
	p{
		color: crimson;
	}
}
```

Les propriétés qui utilisent des valeurs numériques peuvent être préfixées avec min- ou max- afin de tester une plage de valeur :

Cible les écrans dont la largeur est supérieure ou égale à 1024px

```
@media screen and (min-width: 1024px){ ... }
```

Cible les écrans dont la hauteur est inférieure ou égale à 500px

```
@media screen and (max-height: 500px){ ... }
```

Il existe également deux mots clés not et only qui permettent un autre type de restriction. Le mot clé not permet d'inverser la valeur d'une règle ou propriété. Le mot only n'a d'intérêt que pour masquer la requête aux anciens navigateurs qui ne reconnaissent que le media et pas les features. En effet, le mot clé only qui apparaît avant le mot clé screen n'est pas reconnu par ces derniers.

Cible les écrans monochromes

```
@media only screen and (not color){
	html{ color: black; }
}
```

6.3.2. Applications

Il reste donc à préciser l'utilisation des propriétés qui nous intéressent directement, celles concernant

les dimensions des écrans sur lesquels nos sites sont affichés. Nous allons donc voir comment utiliser les différentes options proposées pour cibler les règles en fonction des dimensions d'écrans. Attention, ce sujet peu paraître simple mais un grand nombre de paramètres entrent en compte, n'oubliez pas nous avons « forcé » le navigateur des appareils mobiles à annoncer une taille cohérente en fonction de sa dimension avec la balise meta viewport ou la règle @viewport.

Nous pouvons donc poser une base commune dans notre feuille de styles, puis apporter certaines améliorations pour des tailles d'écrans plus importantes. Dans cet exemple nous modifions le nombre de colonnes des éléments article pour les écrans de plus de 1 024 px de large (figure 6.10).

```
article{
	column-count: 2;
}
@media screen and (min-width:1024px){
	article{
		column-count: 3;
	}
}
```

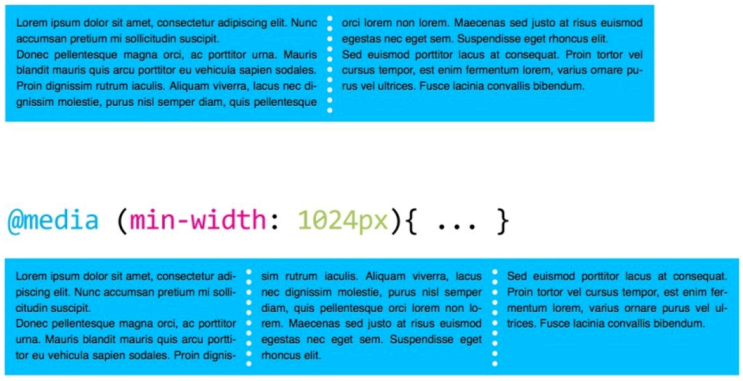


Fig. 6.10 Les medias-queries sont appliquées si la largeur de l'écran est d'au moins 1 024 px

La démarche inverse est bien entendu possible, nous pouvons spécifier des styles pour une taille maximale d'écran. Supprimons par exemple le nombre de colonnes sur les petits écrans (figure 6.11).

```
@media screen and (max-width:420px){
	article{
		column-count: 1;
	}
}
```

Fig. 6.11 Les medias-queries sont appliquées si la largeur de l'écran est au plus 420 px

Il est également possible de cibler une résolution minimum pour traiter le cas particulier des écrans haute densité :

```
@media screen and (min-resolution:250dpi){ ... }
```

Ou encore de cibler le ratio de pixels (rapport entre le nombre de pixels physiques et le nombre de pixels CSS[2]) même si cette fonctionnalité n'est pas dans la spécification mais implémentée dans les principaux navigateurs mobiles.

```
@media screen and (min-device-pixel-ratio: 1.5){ ... }
```

Attention, bien que ces solutions paraissent simples à mettre en place, elles n'en demeurent pas moins risquées. Il est facile de tomber dans l'excès et de commencer à cibler tel ou tel appareil grâce à ces techniques. Imaginez par exemple que demain sorte une montre de 1.5" intégrant un écran HD avec les mêmes caractéristiques que l'appareil ciblé, il y a fort à parier que le résultat ne sera pas optimal.

Media-queries	9.0	3.5	4.0	3.1	9.5	3.2	10.0	2.1

Fig. 6.12 Tableau de compatibilité des media-queries

Le module Media Queries du W3C est en Canditate Recomandation. L'implémentation est donc bonne pour les navigateurs récents, seul IE à reçu cette capacité tardivement. Il est cependant possible de simuler ce comportement avec du JavaScript, la librairie Respond.js effectue cette tâche avec brio. Notez également que certaines des features comme device-pixel-ratio doivent êtres préfixées car elles ne font pas partie de la spécification, et devraient d'ailleurs évoluer.

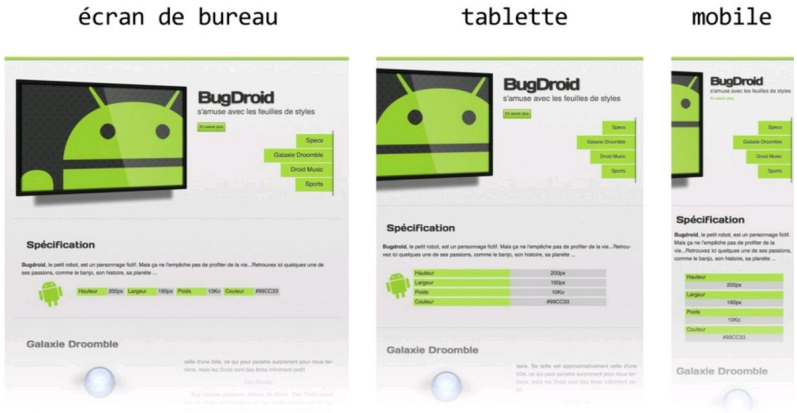


Fig. 6.13 Les différentes versions du site support réalisé avec les media-queries

6.4. Un site optimisé pour tous

Avec la démocratisation du haut débit, nous avons vu exploser le nombre de sites utilisant les technologies gourmandes ou des pages peu optimisées. Il n'est pas rare de voir des sites effectuant une centaine de requêtes HTTP pour l'affichage de la page d'accueil. Même si l'on nous vante un débit 3G+ (HSPDA) à 900 Ko/s en théorie, force est de constater qu'en dehors des grandes agglomérations, le signal est de moindre qualité et ces pages, un peu trop chargées ou mal conçues, sont très lentes à s'afficher.

Il est donc préférable d'essayer de ne pas céder à la facilité de se dire que le poids des pages et l'optimisation sont secondaires dans un réseau « haut débit ». Les moteurs de recherche intègrent d'ailleurs ce paramètre dans leur référencement.

Ceci nous amène à réfléchir à une démarche efficiente dès le début du développement de notre site web. Puisque celui-ci est destiné à être visionné sur un grand nombre de médias hétérogènes, il est important de placer l'accessibilité en bonne place dans le processus de conception tout en limitant au maximum les ressources réseaux devant être utilisées.

Pour cela, voici quelques bonnes pratiques qui sortent du cadre exclusif de CSS :

- Commencer par développer un design de site commun à tous.
- Utiliser le *viewport* pour forcer les terminaux mobiles à se considérer tels qu'ils sont.
- Conditionner l'affichage d'éléments à la spécificité du média avec les *media-queries* :
    - Amélioration progressive
    - Afficher ou ré-ordonner les éléments
    - Ajouter les effets graphiques adaptés
    - Dégradation gracieuse
    - Éviter les éléments côte à côte
    - Adapter votre site à la navigation sans souris
    - Offrir une lisibilité maximale
- Gagner de la place dans les transferts réseaux :
    - Minimiser le nombre de requêtes HTTP
    - Unifier le JS
    - Unifier le CSS

- Utiliser les *sprites* d'images
- Optimiser les images (utiliser SVG quand c'est possible)
- Limiter les utilisations d'AJAX inutiles
- Minimiser le JS
- Minimiser le CSS
- Utiliser les technologies de compressions serveurs

Ce qu'il faut retenir...

Réaliser un site pour tout le monde sans le réécrire pour chacun n'est pas une chose facile. Nous venons de voir que CSS3 nous donne les moyens de réussir ce défi en adaptant le design aux capacités du média sur lequel le site est diffusé. Il paraît impensable de réaliser un site sans prévoir que celui-ci sera visionné sur un mobile par exemple. Ces mobiles sont en constante évolution. Si les iPhones sont assez simple à prendre en compte de part leur exclusivité, les mobiles Android et Windows phones ont presque autant de tailles d'écrans que de modèles existants. Il faut donc user et abuser du responsive web design tout en restant focaliser sur la principale question : nos données sont-elles accessibles rapidement ? Pour cela, il faut donc penser différemment et intégrer cette notion dès la conception même du site.

Nous allons maintenant aborder un tout autre aspect du langage, la mise en mouvement de votre design. Ces fonctionnalités d'animation sont d'ailleurs déjà disponibles sur les navigateurs des principaux systèmes mobiles.

Notes

[1] Cette récente notion pourrait être traduite en français par « Design adaptatif » ou « Design réactif ».

[2] http://www.quirksmode.org/blog/archives/2010/04/a_pixel_is_not.html

Transitions et animations

Objectifs

Jusqu'à présent, de nombreux essais ont été effectués pour dynamiser une page web et interagir avec l'utilisateur, grâce aux possibilités offertes par CSS2.1. Cela se limite principalement à l'utilisation des pseudo-classes :hover, :focus ou encore :active, l'exemple typique étant le changement de l'image d'arrière-plan d'un lien au sein d'un menu. Des solutions originales ont pu être trouvées mais toutes restent terriblement statiques. Dans ce chapitre, nous verrons ensemble ce que nous apporte CSS3 dans ce domaine.

7.1. Du survol à l'animation

CSS2.1 n'étant donc pas satisfaisant en ce qui concerne le dynamisme de la page web, il fallait trouver autre chose. Une solution est apparue avec l'utilisation de JavaScript. Ce langage, disponible dans les navigateurs nous permet entre autres de modifier les propriétés CSS d'un élément du DOM à la volée. Cela reste cependant compliqué et le code est souvent entièrement à réécrire pour la compatibilité multi-navigateurs. La démocratisation des *frameworks* JavaScript et de leur interopérabilité, ainsi que les améliorations en termes de performances JavaScript des navigateurs, ont fait que les API (*Application Programming Interface*) d'animations se sont imposées dans le domaine. Notons par exemple jQuery qui permet d'animer de nombreux objets de la page de façon très fluide et dont la prise en main par des débutants est aisée.

Cependant, bien que ces *frameworks* soient effectivement beaucoup plus facile d'accès que le JavaScript natif, ils reposent tout de même sur ce langage de programmation avec ses bugs et ses contraintes. De plus, les animations générées par ces fonctions ne font que modifier (un très grand nombre de fois) les valeurs des propriétés CSS des éléments du DOM !

Pourquoi alors le CSS lui-même ne permettrait-il pas de modifier dans le temps les valeurs de ses propriétés ? Le JavaScript ne servirait plus qu'à la gestion événementielle de ces animations. Les pseudo-classes telles que :hover, :active, :checked, etc. pourraient également être la source de ce déclenchement. C'est donc exactement ce que proposent les deux modules de CSS3 que nous allons parcourir : effectuer des transitions ou animer de manière plus complexe des propriétés entre deux états CSS. Libre à nous ensuite de déclencher ces transitions/animations avec des événements CSS, avec une temporisation ou par le biais de JavaScript.

Nous verrons tout d'abord comment utiliser les transitions pour passer d'un état à un autre, puis nous essaierons d'appréhender les propriétés d'animations qui, avec votre imagination, vous permettront de vous affranchir des *frameworks* JavaScript ou encore du très répandu plugin Flash.

7.2. Les transitions

Une transition CSS est un ensemble de propriétés qui définissent une interpolation temporelle d'une

ou plusieurs propriétés CSS entre deux états de celles-ci. Pour faire plus simple, une transition anime le changement d'état entre deux valeurs différentes d'une propriété CSS. Comme un exemple parle toujours plus qu'un long discours et même si l'interpolation temporelle n'est pas facile à reproduire sur le papier, voici le concept présenté par un exemple simple.

Lors du survol d'une image, celle-ci disparaît progressivement. Effectuons alors une transition sur la propriété opacity lors du survol de celle-ci qui devient le nouvel état.

```
<div><img src="Bugdroid.jpg" alt="CoolDroid" /></div>
```

```
div img{
	opacity: 1;
}
div img:hover{
	opacity: 0;
	transition: opacity 2s linear;
}
```

Fig. 7.1 L'élément est rendu progressivement transparent

Dans la figure 7.1, nous utilisons la propriété raccourcie transition qui modifie la propriété opacity en deux secondes grâce à une interpolation linéaire, mais nous allons voir tout cela en détail.

7.2.1. transition-property

La propriété transition-property permet de définir la propriété qui sera animée lors de la transition. Les valeurs possibles sont :

- all : toutes les propriétés animables (voir la liste plus bas).
- none : aucune transition.
- Une ou plusieurs propriétés animables séparées par une virgule.

```
transition-property: color, width;
```

Attention, certaines implémentations limitent l'exécution d'une transition à des éléments dont l'unité ne change pas. Par exemple il n'est pas possible sur Chrome d'animer une marge de 10 % à 800 px car les unités diffèrent. Il en est de même pour les valeurs fixées à auto, il existe cependant des astuces pour obtenir le résultat escompté. Citons par exemple l'utilisation des transformations 2D à l'aide de la valeur scale (échelle), ou encore la propriété max-height en lieu et place de height.

7.2.2. transition-duration

Cette propriété définit tout simplement la durée totale de la transition exprimée en secondes. La

valeur par défaut est zéro seconde ce qui ne provoque aucune transition lors du changement d'état. Voici la syntaxe pour une transition d'une durée de quatre secondes.

```
transition-duration: 4s;
```

7.2.3. transition-timing-function

Cette propriété définit la progression ou l'évolution de la vitesse de transition. Une fonction mathématique va permettre le calcul des différentes valeurs interpolées lors de la transition, c'est cette fonction que nous devons choisir pour notre transition. Il existe des raccourcis vers des fonctions prédéfinies mais aussi des fonctions permettant de définir plus finement cette évolution. Deux types d'évolution sont disponibles: l'interpolation par courbe de Bézier et l'interpolation par palier, bien que celle-ci soit encore peu déployée. Commençons par décrire la première.

cubic-bezier

La fonction cubic-bezier(x1,y1,x2,y2) permet de définir une courbe de Bézier décrivant la portion de transition effectuée en fonction du temps écoulé (figure 7.2). Cette courbe est décrite par quatre points :

- le 1er point est le point de départ.
- le 2ème point définit la direction de la courbe partant du 1er point.
- le 3ème point définit la direction de la courbe arrivant au 4ème point
- le 4ème point est le point d'arrivée.

Nous pouvons représenter cette courbe et ses points la définissant dans un diagramme ayant pour ordonnée le pourcentage de transition effectuée (ramené à 1) et en abscisse le temps écoulé, lui aussi ramené à 1.

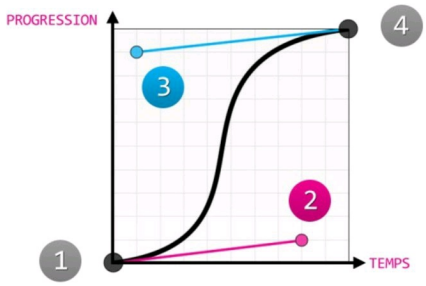


Fig. 7.2 Les quatre points qui définissent une courbe de Bézier

Dans cet exemple, pour un temps écoulé de 30 % (0,3), la transition est effectuée à 10 % et à 50 % du temps la transition en est à 65 %. Cela provoque donc un effet d'accélération de la transition après un départ lent. Inversement, à 70 % du temps la transition est effectuée à 90 %, on a donc un ralentissement de la transition vers la fin.

Nous verrons par la suite de nombreux exemples de courbes de Bézier.

Cette courbe dont la fonction est cubic-bezier(0.8, 0.1, 0.1, 0.9) se définit en CSS3 par les coordonnées dans le repère unitaire des points P2 et P3, de cette façon (figure 7.3) :

```
cubic-bezier( xP2 , yP2 , xP3 , yP3 )
```

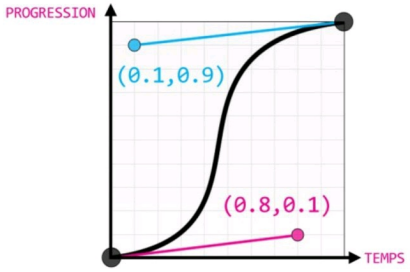


Fig. 7.3 Les deux points utiles

La spécification précise que les valeurs *x* doivent être comprises entre 0 et 1 (cela paraît logique, nous ne savons pas encore remonter le temps...) alors que les valeurs *y* sont libres. Cependant l'implémentation des navigateurs n'est pas uniforme et certains moteurs n'acceptent pas les valeurs *y* négatives. Cette particularité est pourtant intéressante car, en décrivant une courbe qui sort de son carré de définition, nous pouvons obtenir une transition qui passe par des valeurs extérieures à ses bornes. Par exemple, une transition de marge de 0 px à 100 px pourrait passer par une étape à –10 px ou à 130 px. Nous verrons ce cas dans les exemples qui suivent.

Des mots-clés ont été définis pour représenter les courbes les plus couramment employées, les voici avec les valeurs de courbes de Bezier équivalentes :

Ease

```
transition-timing-function: ease; (valeur par defaut)
transition-timing-function: cubic-bezier(0.25, 0.1, 0.25,1.0)
```

Ease-in

```
transition-timing-function: ease-in;
transition-timing-function: cubic-bezier(0.42, 0, 1.0,1.0)
```

Ease-out

```
transition-timing-function: ease-out;
transition-timing-function: cubic-bezier(0, 0, 0.58, 1.0)
```

Ease-in-out

```
transition-timing-function: ease-in-out;
transition-timing-function: cubic-bezier(0.42, 0, 0.58,1.0)
```

Linear

```
transition-timing-function: linear;
transition-timing-function: cubic-bezier(0, 0, 1.0,1.0)
```

Il est bien entendu possible de définir nos propres courbes :

```
transition-timing-function: cubic-bezier(0.1, 1.2, 0.6, 1.1 );
```

La figure 7.4 montre comment les fonctions modifient le déroulement d'une transition simple. Nous appliquons ici une transition sur la largeur d'un bloc de 0 à 100 % en quatre secondes.

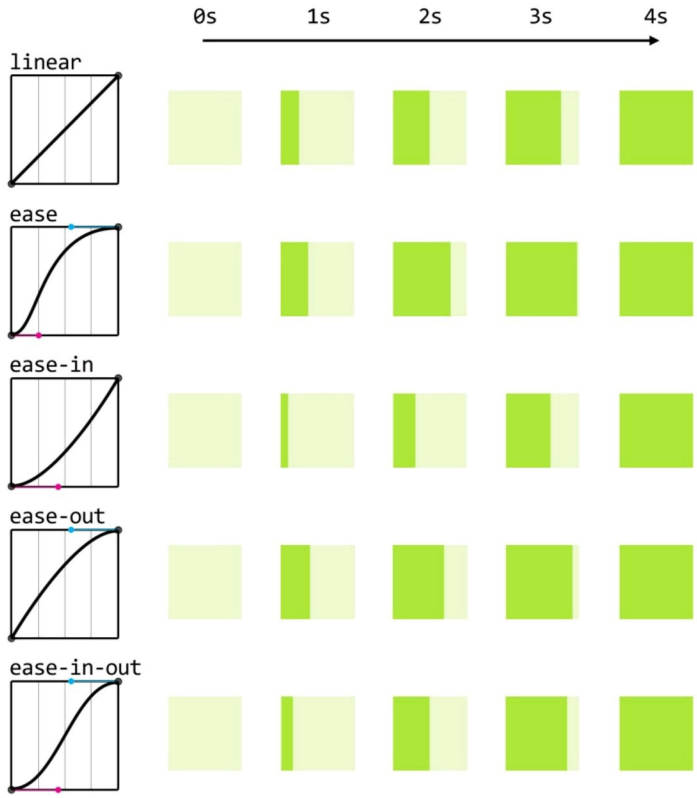


Fig. 7.4 Les différentes fonctions d'accélération

steps

Pour terminer sur les fonctions de transition, il est également possible de définir l'ajustement d'une transition à l'aide d'une fonction par palier. Nous n'utiliserons pas cette fonction dans notre démo mais en voici le principe.

Cette fonction nommée steps() permet de convertir une transition linéaire en paliers. La transition est alors découpée en *n* phases. Par exemple, la fonction steps(4) permet de générer trois états intermédiaires entre l'état initial et l'état final. Il n'y a aucune animation entre les états. Les mots clés start ou end peuvent être ajoutés en deuxième paramètre et nous indiquent si l'animation débute instantanément ou après le premier palier (figure 7.5). Si ce deuxième paramètre est omis, c'est la valeur end qui est utilisée. Nous verrons dans la partie sur les animations comment tirer parti de cette propriété.

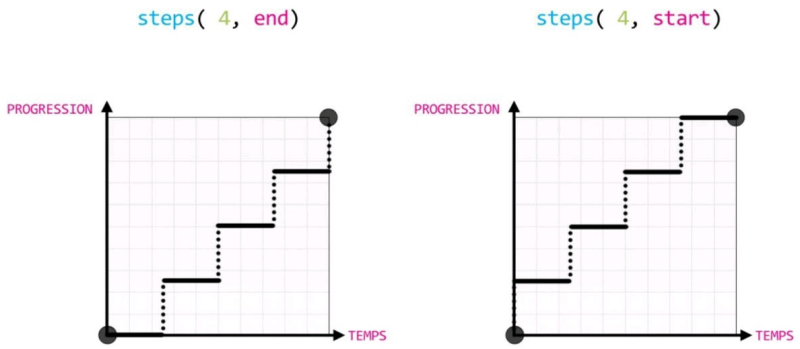


Fig. 7.5 Fonctions de transition par palier

7.2.4. transition-delay

Il reste une propriété simple relative aux transitions, il s'agit de la propriété transition-delay. Cette propriété permet un départ retardé de la transition. Si la valeur précisée est positive, alors le départ de la transition sera décalé d'autant. Si cette valeur est négative, l'animation de la transition débute immédiatement mais depuis l'état interpolé correspondant au delta de temps indiqué en valeur négative. La durée totale de l'animation est diminuée de cette valeur. Par exemple, dans le cas d'une interpolation de distance sur une transition de quatre secondes avec transition-delay: -2s, l'animation va débuter immédiatement mais du milieu de la transition et non pas du point de départ de celle-ci. L'animation totale dure alors deux secondes (figure 7.6).

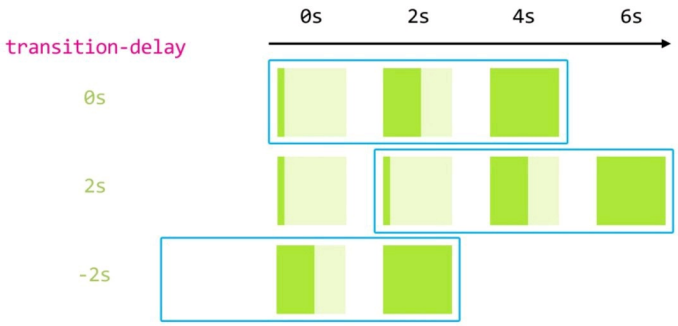


Fig. 7.6 La propriété transition-delay retarde l'animation.

7.2.5. transition

Comme pour beaucoup de propriétés composées en CSS, il existe un raccourci qui permet de renseigner la plupart des valeurs avec une seule instruction. Il faut donc saisir les différentes valeurs séparées par un espace (la première valeur temporelle correspond à la durée, la deuxième au délai).

```
transition: width  2s  ease-in-out  3s;
```

propriété durée accélération délai

Fig. 7.7 Syntaxe de la propriété transition

7.2.6. Prise en charge

Comme toutes les améliorations récentes de CSS3, les transitions ne sont pas gérées par tous les navigateurs, comme nous pouvons le voir sur le tableau de compatibilité suivant. Fort heureusement, les navigateurs non compatibles ignoreront simplement ces règles qu'ils ne connaissent pas. Il est donc important de commencer par définir votre design initial et d'appliquer une amélioration progressive en intégrant les transitions. Elles doivent valoriser votre contenu et accroître l'expérience utilisateur tout en étant subtilement intégrées à vos pages.

Transitions CSS	10.0 -ms-	4.0 -moz-	4.0 -webkit-	3.1 -webkit-	10.5 -o-	3.2 -webkit-	10.0 -o-	2.1 -webkit-

Fig. 7.8 Tableau de compatibilité des transitions CSS

Pour finir sur les transitions, nous vous proposons le tableau 7.1 tiré du site du W3C qui est aussi valable pour les animations qui vont suivre et qui présente les propriétés qui sont théoriquement animables. Attention cela dépend beaucoup de l'implémentation réalisée dans les navigateurs.

Tab. 7.1

Propriétés animables		
background-color	font-size	outline-width
background-image (dégradés uniquement)	font-weight	padding-bottom
background-position	height	padding-left
border-bottom-color	left	padding-right
border-bottom-width	letter-spacing	padding-top
border-color	line-height	right
border-left-color	margin-bottom	text-indent
border-left-width	margin-left	text-shadow
border-right-color	margin-right	top
border-right-width	margin-top	vertical-align
border-spacing	max-height	visibility
border-top-color	max-width	width
border-top-width	min-height	word-spacing
border-width	min-width	z-index
bottom	opacity	zoom
color	outline-color	
crop	outline-offset	

7.2.7. À vous de jouer !

Pour mettre en œuvre ces notions, nous allons réaliser une petite animation en intégrant petit à petit les différents éléments qui composent la transition CSS. Encore une fois n'oubliez pas que les exemples donnés ici ne sont pas préfixés. L'animation que nous allons reproduire correspond au menu de navigation du site de démonstration: lors du survol, l'élément sélectionné se déplace progressivement vers la gauche. Dans un deuxième temps les autres éléments du menu subissent un fondu qui les atténue (figure 7.9).

Fig. 7.9 Menu du site de démonstration

Pour réaliser cette démonstration, nous avons besoin d'une structure de menu dont voici le HTML à intégrer dans le body d'une page HTML5 :

HTML

```
<nav>
	<ul>
		<li><a href="#secondPart">Specs</a></li>
		<li><a href="#thirdPart">Galaxie Droomble</a></li>
		<li><a href="#fourthPart">Droid Music</a></li>
		<li><a href="#fifthPart">Sports</a></li>
	</ul>
</nav>
```

Nous ajoutons progressivement le design initial de notre menu :

CSS

```
ul li{
	display: block;
	text-align: right;
	margin: 4px 0;
}
```

Puis les liens à l'intérieur des éléments de liste :

```
ul li a{
	padding: 10px 30px;
	display: inline-block;
	color: #333;
	background: hsl(80,68%,60%);
	text-decoration: none;
}
```

Un peu de design :

```
ul li a{
	....
	text-shadow: 0 1px 0 rgba(255,255,255,.8);
	border-top: 1px solid rgba(255,255,255,.4);
	border-bottom: 1px solid rgba(0,0,0,.1);
}
```

Il nous faut maintenant intégrer la transition. Lors du survol du lien, mais également lorsque celui-ci a

le focus, nous allons agrandir le padding à droite :

```
ul li a:hover, ul li a:focus{
	padding-right: 60px;
}
```

Ajoutons la transition (figure 7.10) :

```
ul li a:hover,ul li a:focus{
	...
	transition-property: padding;
	transition-duration: 1s;
}
```

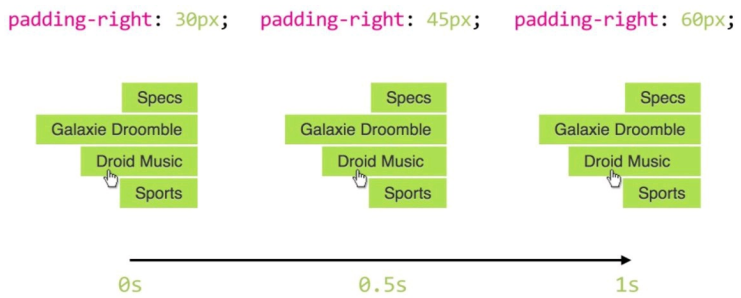


Fig. 7.10 Lors du survol, la marge interne gauche est modifiée.

Et voilà, notre élément de menu se déplace bien de manière fluide vers la gauche lors du survol. Qu'en est-il du retour à l'état normal ? Il n'est pas animé. Deux solutions s'offrent à nous pour gérer ce cas :

- soit la déclaration de transition est déplacée dans le bloc ul li a. L'héritage va alors appliquer cette transition dans les deux cas ;
- soit deux transitions différentes sont utilisées. C'est d'ailleurs ce que nous vous proposons, en diminuant la durée de la transition pour le retour à l'état normal :

```
ul li a{
	...
	transition-property: padding;
	transition-duration: 0.3s;
}
```

Le menu reprend sa position initiale effectivement plus rapidement qu'il ne l'a quittée. Bien que le rendu soit satisfaisant, nous allons essayer d'améliorer l'effet visuel, grâce à la propriété transition-timing-function afin de préciser de quelle manière va progresser la transition. Il nous faut donc une courbe qui décrive une accélération rapide puis un ralentissement sur la toute fin de la transition. Essayons avec le mot clé ease-out :

```
ul li a{
	...
	transition-property: padding;
	transition-duration: 0.3s;
	transition-timing-function: ease-out;
}
```

C'est mieux ! Mais pas encore suffisant. Définissons donc notre propre courbe (figure 7.11) :

```
ul li a{
	...
	transition-timing-function: cubic-bezier( 0.5, 1, 0.5, 0.8);
}
```

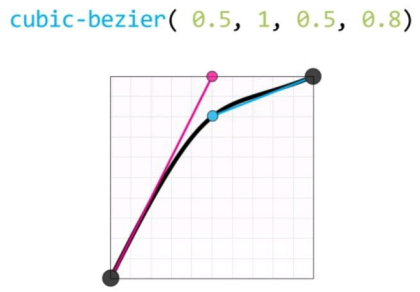


Fig. 7.11 Courbe personnalisée

C'est beaucoup plus réaliste comme ça. Si vous utilisez un navigateur compatible (comme Firefox), vous pouvez même essayer une courbe comme celle-ci avec des coordonnées en dehors des limites [0-1] pour l'axe de la progression de l'animation (figure 7.12). Cette courbe va simuler un léger rebond :

```
transition-timing-function: cubic-bezier( 0.5, 2, 0.5, 0.8);
```

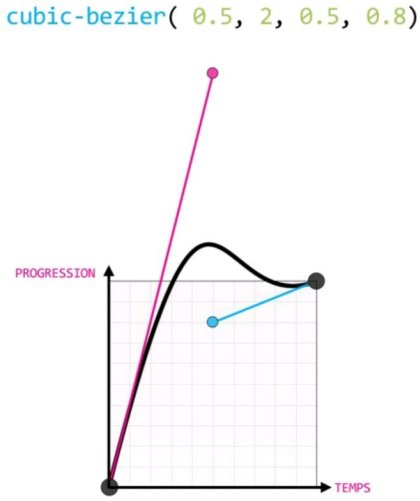


Fig. 7.12 Courbe personnalisée hors-bornes

Pour simplifier l'écriture de ces propriétés, nous pouvons utiliser la syntaxe raccourcie, nos transitions précédentes deviennent donc :

```
ul li a{
	...
	transition: padding 0.3s cubic-bezier( 0.5, 1, 0.5, 0.8);
}
ul li a:hover, ul li a:focus{
	...
	transition: padding 1s ease;
}
```

Pour parfaire le rendu visuel, nous ajoutons l'effet très intéressant d'atténuation des autres éléments du menu. Pour cela nous faisons appel à une astuce utilisant le nouveau sélecteur CSS3 :not. En voici l'idée principale :

« Nous devons sélectionner tous les a *dont le* li *n'est pas survolé*
...mais dont le ul *parent est survolé. »*

De cette façon, nous n'interférons pas avec le a:hover classique qui permet la transition initiale. On obtient :

```
ul:hover li:not(:hover) a{
	background: hsl(80,30%,80%);
}
```

Il nous reste à appliquer la transition sur le changement de couleur. Le raccourci de la propriété transition peut prendre plusieurs transitions en paramètres, il suffit de les séparer par des virgules :

```
ul li a{
	...
	transition: padding 0.3s cubic-bezier( 0.5, 1, 0.5, 0.8) , background 1s;
}
ul li a:hover, ul li a:focus{
	...
	transition: padding 1s ease, background 1s;
}
```

Attention, même si les deux transitions sont les mêmes, il faut dupliquer l'animation du background dans la transition du :hover sinon, elle serait ignorée car la définition de la transition du second bloc écrase celle héritée par le premier.

Nous pouvons maintenant terminer l'animation en ajoutant une petite barre simulant l'insertion des menus dans l'écran. Pour cela nous allons générer du contenu invisible à l'aide du pseudo-élément ::after puis nous le mettons en forme avec un petite ombre à gauche.

```
ul::after{
	content: '';
	position: absolute;
	top: -10px;
	right: 0;
	width: 10px;
	height: 198px;
	box-shadow: -5px 0 5px -5px rgba(0,0,0,.7) inset;
}
```

7.3. Les animations

Nous venons de voir qu'il était possible de faire des transitions sur les éléments de nos pages web. Cela nous restreint tout de même à des animations simples, les transitions ayant comme limitation de n'avoir qu'un état de début et de fin entre lesquels l'animation a lieu. Le contrôle des transitions est lui aussi très limité : le déclenchement (sans utilisation de JavaScript) se limite presque exclusivement aux interactions de souris ou de focus. C'est donc surtout pour avoir un contrôle plus abouti que les animations CSS ont été proposées.

Avec les animations CSS, il est possible de définir autant d'états différents que nécessaires. Ces états sont appelés des *keyframes* qui se traduisent littéralement par *images clés* mais dont la traduction *états clés* nous paraît plus appropriée. Il est également possible de préciser la répétition de l'animation ainsi que son état final et son délai de déclenchement. Mais voyons cela de plus près.

7.3.1. @keyframes

C'est cette @-règle qui va permettre de définir les différentes étapes de l'animation. Elle est pour l'instant encore préfixée pour Webkit, Gecko et IE10 mais son implémentation est assez bonne dans les navigateurs concernés. Les étapes de l'animation vont être identifiées par un pourcentage ou l'un des mots clés from qui équivaut à 0 % et to qui équivaut à 100 %. Pour chaque étape, comme lors d'une transition simple, nous allons définir les règles qui subissent une modification.

Il nous reste ensuite à donner un nom à cette @keyframes pour pouvoir l'associer avec une animation. Voici l'exemple d'une règle définissant les étapes d'une transition classique :

```
@keyframes hide{
	0%{
		opacity:1;
	}
	100%{
		opacity:0;
	}
}
```

Dans cet exemple, nous définissons deux *états clés* permettant la transition entre l'opacité totale et la transparence totale. Cette animation n'est toutefois pas encore appliquée.

Si la valeur de l'une des propriétés à interpoler n'est pas présente à une étape, l'interpolation sera faite entre les valeurs connues.

```
@keyframes hide{
	0%{
		opacity: 1;
		background-color: red;
	}
	50%{
		background-color: blue;
	}
	100%{
		opacity: 0;
		background-color:yellow;
	}
}
```

Cet exemple introduit donc deux phases de changement de couleur associées à un fondu régulier étalé sur la totalité de l'animation.

Il est également possible de définir un même bloc de définitions pour plusieurs états en séparant les valeurs de ces états par des virgules :

```
@keyframes bounce{
	0%, 100%{
		margin-top: 0;
	}
	50%{
		margin-top: 100px
	}
}
```

7.3.2. animation-name, animation-duration

Pour appliquer une animation à un élément, il suffit de lui définir la propriété animation-name en précisant le nom de la @keyframes à utiliser puis de lui indiquer une durée avec la règle animation-duration. La propriété animation-name peut prendre comme valeur un ou plusieurs noms d'animation ou bien encore la valeur none pour par exemple annuler un héritage dans le cas d'utilisation des pseudo-classes. La propriété animation-duration quant à elle n'accepte que des durées positives.

```
div img{
        animation: hide;
        animation-duration: 5s;
}
```

L'effet obtenu n'est pas très impressionnant. Contrairement aux transitions, une animation débute lors du chargement de la page et non pas uniquement lors d'un changement d'état. Mis à part cette particularité, l'animation n'apporte ici pratiquement aucune amélioration par rapport à une transition. Mais ce n'est qu'un début, voyons ce que nous pouvons ajouter comme paramètres à notre animation pour la rendre plus intéressante.

7.3.3. animation-iteration-count

La propriété animation-iteration-count définit le nombre de répétitions de l'animation. C'est une chose impossible à réaliser avec les transitions. Il existe même le mot-clé infinite qui permet de répéter l'animation sans fin.

```
div img{
        ...
        animation-iteration-count: infinite;
}
```

C'est déjà plus intéressant ! En revanche, le retour à l'opacité 0 est instantané ce qui n'est pas du plus bel effet. Mais bien entendu il y a une propriété qui a été prévue pour pallier ce problème.

7.3.4. animation-direction, animation-delay

La propriété animation-direction permet de définir si la répétition de l'animation se déroule normalement ou si elle alterne avec l'animation inverse. Les deux valeurs possibles pour cette propriété sont normal et alternate, normal étant la valeur par défaut.

```
div img{
            ...
        animation-iteration-count: infinite;
        animation-direction: alternate;
}
```

Nos images s'animent bien dans un cycle sans fin en apparaissant puis disparaissant. Comme pour les transitions, il est également possible de définir un départ retardé de l'animation avec la propriété animation-delay qui prend comme valeur une durée. Si la valeur est positive, alors l'animation débutera après *n* secondes, si la valeur est négative, l'animation débutera instantanément mais à partir de la position qu'elle aurait du avoir après *n* secondes.

Nous pouvons aussi évoquer la présence dans les implémentations de la propriété animation-play-state dont les valeurs running et paused permettent de contrôler l'état d'une animation. Cette propriété a semble-t-il été suspendue de la spécification bien qu'elle soit présente dans les principaux navigateurs.

7.3.5. animation-fill-mode

Par défaut, une animation ne modifie pas les propriétés sur lesquelles elle s'applique avant le début (durant le animation-delay) et après la fin de celle-ci. La propriété animation-fill-mode permet de modifier ce comportement. Les valeurs possibles sont :

- none : comportement par défaut
- backwards : applique les propriétés définies dans l'état 0% avant le début de l'animation, y compris le délai.
- forwards : applique les propriétés définies dans l'état final de l'animation une fois l'animation terminée (ce peut être 100 % ou 0 % si la valeur de animation-direction est alternate).
- both : combinaison de forwards et backwards.

Il est donc souvent judicieux d'utiliser le mot clé forwards pour laisser le design identique à l'état correspondant à l'animation terminée.

7.3.6. animation-timing-function

Cette propriété est totalement identique à celle décrite dans le chapitre sur les transitions, à ceci près que la fonction sélectionnée s'applique à chaque portion de l'animation et non à l'animation totale. Il est possible de définir individuellement la fonction d'animation pour chaque phase de l'animation.

Notons tout de même les mots-clés step-start et step-end qui sont des raccourcis pour les valeurs steps(1,start) et steps(1,end). Ils ont pour effet de ne pas effectuer de transition entre deux étapes.

Un exemple amusant de cette propriété avec la fonction steps, que nous n'avons pas détaillée au chapitre précédent, est de faire un dessin animé à l'aide d'une image *sprite*. C'est le sujet de la démonstration qui suit.

7.3.7. animation

Les propriétés d'animation n'échappant pas à la règle, il existe une notation raccourcie dont voici la syntaxe :

```
animation: anim  2s  ease-out  3s  5  alternate;
```

nom | durée | accélération | délai | x5 | direction

Fig. 7.13 Syntaxe de la propriété animation

Les unités de décalage et de durée (secondes : s) sont obligatoires même pour la valeur zéro afin de pouvoir faire la distinction entre les valeurs de temps et de répétition dans la notation raccourcie.

Animations CSS	10.0 -ms-	5.0 -moz-	4.0 -webkit-	4.0 -webkit-	12.0 -o-	3.2 -webkit-		2.1 -webkit-

Fig. 7.14 Tableau de compatibilité des animations CSS

7.3.8. À vous de jouer !

Pour ce cas pratique, nous allons réaliser l'animation du *Droid skater* présente sur le site de démonstration. Cette animation utilise une image telle que celle-ci disponible depuis le site (figure 7.15).

Fig. 7.15 Image utilisée pour l'animation

Le principe est de visualiser un seul *Droid*, puis de déplacer l'image en utilisant des paliers afin d'afficher successivement les différents *Droids*. L'animation n'est donc pas linéaire.

HTML

```
<div>
<img src="bugdroidSkater.png" alt="Droid skater" />
</div>
```

Puis en CSS, la div est dimensionnée à 200 px affichant le premier *Droid* et le reste de l'image est masqué par la déclaration overflow: hidden (figure 7.16) :

CSS

```
div{
	width: 200px;
	overflow: hidden;
}
```

Un peu de design :

```
div{
	...
	background: linear-gradient( rgba( 255, 255, 255, 0.2), rgba( 255, 255, 255, 0));
}
```

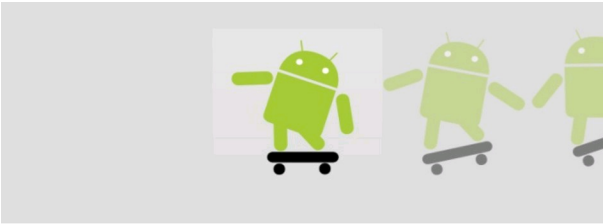


Fig. 7.16 Le reste de l'image est masqué

Enfin, l'image doit être en position:relative; pour pouvoir modifier sa position facilement :

```
div img{
	position: relative;
}
```

Place à l'animation grâce à la règle @keyframes que vous n'oublierez pas de préfixer. Nous programmons deux phases, le déclenchement du saut sur le premier quart de l'animation suivi d'une phase sans mouvement qui simule le roulement. Il est possible d'utiliser les mots clés from et to en lieu et place des 0 % et 100 %. Nous appliquons donc un décalage de –1 000 px pour afficher le dernier *Droid* dans l'étape finale, l'image mesurant 1 200 px de long.

```
@keyframes skate{
	from { left:0; }
	25%  { left:-1000px; }
	to   { left:-1000px; }
}
```

Appliquons cette animation en boucle à l'image :

```
div img{
	position: relative;
	animation: skate 2s infinite;
}
```

L'image se déplace bien, mais on est loin de l'effet « dessin animé ». Utilisons donc la fonction de temporisation pour réaliser notre effet, en définissant cinq paliers. Le décalage se fait alors en six étapes, une pour chaque représentation du *Droid* sur l'image.

```
div img{
	position: relative;
	animation: skate 2s infinite steps(5);
}
```

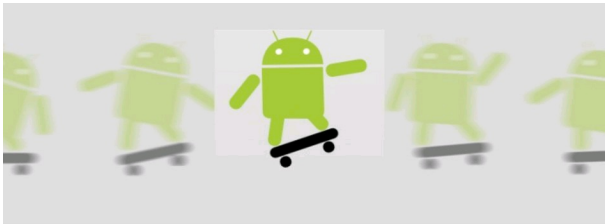


Fig. 7.17 L'image se déplace à la manière d'un dessin-animé

Et voila, si tout se passe bien votre *Droid* doit s'élever de façon synchronisée (figure 7.17).

7.4. Les nouveaux évènements Javascript

Même si le JavaScript ne fait pas partie du programme de cet ouvrage, nous nous devions d'évoquer ces nouveaux évènements qui sont directement liés aux animations et aux transitions CSS. En effet il est possible de détecter la fin d'une transition ou encore la répétition d'une animation grâce aux événements DOM qui sont propagés. Il est donc possible d'intégrer les animations CSS dans un contexte d'application JavaScript sans avoir recours aux fonctions d'animation telles que celles

fournies par jQuery par exemple.

Voici donc un petit exemple de récupération d'un évènement de fin de transition, en JavaScript pur puis en utilisant le Framework JQuery.

Commençons par le HTML, simple comme toujours dans nos exemples :

```
<div id="css3">Bientôt CSS 3</div>
```

Poursuivons avec le CSS (n'oubliez pas de préfixer correctement) :

```
#css3{
	background: crimson;
	padding: 1em;
	font-size: 1em;
	transition: font-size 3s;
}
#css3:hover{
	font-size: 5em;
}
```

La transition est simple, au survol, la taille de la police augmente de façon significative.

Nous allons maintenant intercepter la fin de la transition avec les évènements DOM en JavaScript. Ces évènements, encore expérimentaux, adoptent donc une syntaxe préfixée dont voici les différentes versions pour l'événement qui nous intéresse transitionend (tableau 7.2).

Tab. 7.2

	Version	Propriété CSS	Évènement
Webkit	1	-webkit-transition	webkitTransitionEnd
Firefox	4	-moz-transition	transitionend
IE	10	-ms-transition	MSTransitionEnd
Opéra	10.5	-o-transition	oTransitionEnd

Pour récupérer l'évènement, il faut attacher un guetteur (*listener* en anglais : une fonction de surveillance) à l'élément que l'on veut surveiller. Lorsque l'évènement que l'on écoute se produit, le guetteur déclenche la fonction définie pour cette action. Voici donc le code JavaScript correspondant à cette étape pour le navigateur firefox :

Récupération de l'objet DOM correspondent à la div#css3

```
var css3 = document.getElementById("css3");
```

Ajout du guetteur et définition de la fonction à exécuter lors du déclenchement

```
css3.addEventListener("transitionend", function(event){
	//action à réaliser
}, false);
```

Si vous testez ce code avec un navigateur basé sur Webkit. Remplacez simplement transitionend par webkitTransitionEnd. Cette modification n'est pas très productive mais n'oubliez pas que c'est une

fonctionnalité encore expérimentale. Il existe de nombreuses manières d'améliorer cette prise en charge avec des librairies JavaScript mais ce n'est pas le but de cet ouvrage.

Pour bien nous rendre compte de l'effet, nous allons changer le texte de la div une fois la transition terminée. Dans le code suivant, this désigne l'élément sur lequel on a appliqué le guetteur, c'est donc bien l'élément div :

```
css3.addEventListener("transitionend", function(event){
	this.innerHTML = "CSS 3 est arrivé!";
}, false);
```

Si vous préférez utiliser un *framework*, il est possible d'y intégrer ces nouveaux évènements si ce dernier le permet. Voyons cet exemple avec jQuery pour les trois principaux moteurs :

```
$("#css3").bind("transitionend", function(e){animEnd(this,e) })
		.bind("webkitTransitionEnd",function(e){animEnd(this,e)})
		.bind("MSTransitionEnd",function(e){animEnd(this,e)});
var animEnd = function(obj,e){
	$(obj).text("CSS 3 est arrivé");
}
```

Voilà donc ce qu'il est possible de réaliser en couplant la puissance des transitions CSS3 et la facilité d'utilisation de jQuery. Pour finir, voici la liste des nouveaux évènements disponibles (tableau 7.3) :

Tab. 7.3

Événement	Définition
transitionend	évènement de fin de transition.
animationstart	évènement de début d'animation.
animationiteration	évènement déclenché lors de la répétition d'une animation dont la valeur pour animation-iteration-count est supérieure à 1.
animationend	évènement de fin d'animation.

Ce qu'il faut retenir...

Nous avons pu voir tout au long de ce chapitre que les transitions et animations CSS reprennent le pouvoir sur JavaScript, dont certaines fonctionnalités ont été détournées dans le but de réaliser une présentation dynamique des éléments d'une page web. En effet, qu'elle soit statique ou dynamique, la mise en forme d'un document web est du ressort de CSS.

De nombreux sites d'envergure ont déjà mis en place ces mécanismes en suivant le principe de l'amélioration progressive, ce qui permet d'accroître l'expérience utilisateur au sein des navigateurs compatibles. Si nous pensons que les premiers à tirer parti de ces capacités seront les publicitaires, les designers auront autant de flèches à leurs arcs pour réserver de bonnes surprises aux visiteurs.

Continuons notre découverte de CSS3 avec les transformations géométriques, qui poussent encore plus loin les capacités du design web moderne.

Transformations 2D et 3D

Objectifs

Introduites dans le WD (Working Draft) CSS 3 du W3C suite aux expérimentations de l'équipe Webkit, les transformations CSS ont très rapidement éveillé l'intérêt des designers web et des éditeurs de navigateurs. En effet ces nouvelles propriétés vont nous permettre de modifier l'aspect des éléments du DOM sur nos pages. C'était déjà le cas avec le position:absolute; ou position:relative; qui nous permet de déplacer l'élément sur un calque virtuel, mais la nouvelle propriété transform va nous offrir beaucoup plus en introduisant des modifications d'ordre géométrique. C'est ce que nous détailler dans ce chapitre.

8.1. Les transformations 2D

« *Et si j'inclinais ce bandeau de titre, ce serait pas mal non ?* » Vous vous êtes sûrement posé cette question, et vous vous êtes aussi sûrement rué vers votre logiciel de traitement d'image pour en produire le résultat, ou bien alors vous êtes un gourou de JavaScript et de canvas. Et bien ce temps est révolu. Enfin presque, car à partir d'IE9 tous les navigateurs implémentent correctement les transformations 2D. Et nous allons le voir, incliner un élément quelconque de la page est un jeu d'enfants. De plus, le fait d'utiliser CSS pour effectuer des modifications d'ordre graphique permet à nos pages de garder toute leur sémantique.

Les dernières versions de nos navigateurs ont même intégrés la troisième dimension à cet outil spectaculaire. Nous aborderons les spécificités de cet aspect dimensionnel un peu plus loin mais sachez déjà que la syntaxe est la même.

Il est bon de noter qu'aucun navigateur n'utilise à ce jour la syntaxe officielle qui n'est toujours pas finalisée, les préfixes sont donc obligatoires (-moz-,-webkit-,-ms,-o-...).

8.1.1. transform

La propriété transform permet donc de transformer un élément du DOM dans l'espace. Les transformations applicables sont : la translation, la rotation, l'inclinaison et l'échelle. Nous allons détailler chacune de ces transformations et voir que nous pouvons bien entendu en utiliser plusieurs simultanément. Comme beaucoup de notation CSS, il existe une notation raccourcie permettant le cumul de plusieurs règles.

Par défaut, les éléments subissant une transformation se comportent comme s'ils étaient positionnés en position:relative; car l'espace « liberé » par leur transformation n'est pas comblé et l'élément transformé se retrouve dans un « calque » superposé au reste de la page.

Dans la version 2D, la propriété transform peut prendre les valeurs suivantes sous forme de fonctions avec leurs paramètres.

translateX, translateY, translate

Les fonctions translate(), translateX() et translateY() effectuent une translation verticale, horizontale ou combinée (figure 8.1). Cette transformation n'est vraiment utile qu'en association avec une autre transformation car le résultat obtenu est équivalent à un déplacement en position relative.

```
transform: translateX(150px);
transform: translateY(2em);
transform: translate(150px,2em);
```

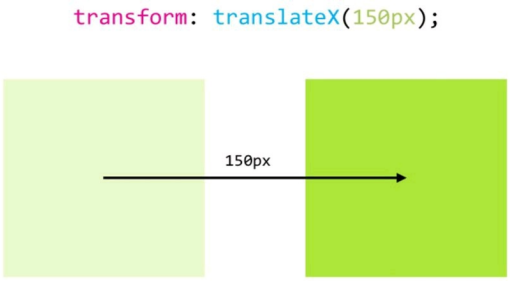


Fig. 8.1 Rendu d'une translation

rotate

La fonction rotate() est l'une des transformations les plus attendues, elle permet une rotation dans le plan de la page dans le sens horaire (figure 8.2). L'utilisation de l'unité deg est alors toute désignée. La nouvelle unitée turn semble intéressante mais son implémentation est encore très limitée.

```
transform: rotate(45deg);
```

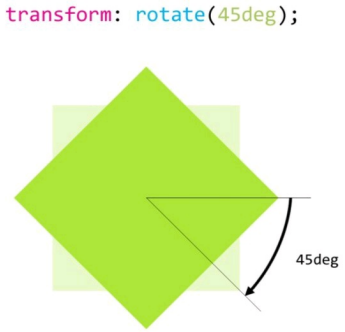


Fig. 8.2 Rendu d'une rotation

skewX, skewY

Les fonctions skewX et skewY correspondent à l'inclinaison (figure 8.3). Elles peuvent se rendre fort utiles dans la simulation de design du style *2D iso*. Nous pouvons effectuer une inclinaison horizontale ou verticale.

```
transform: skewX(30deg);
transform: skewY(45deg);
```

```
transform: skewX(45deg);
```

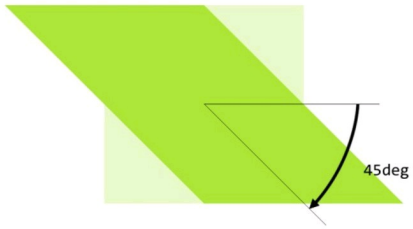


Fig. 8.3 Rendu d'une inclinaison

scaleX, scaleY, scale

La fonction scale() correspond à l'agrandissement (figure 8.4). Le changement d'échelle d'un élément peut être dangereux. En effet, le texte vectoriel se comporte correctement au contraire des images qui sont pixellisées ou mal interpolées. Il est néanmoins très utile associé aux transitions dans des effets de survol par exemple. Une échelle de 1 ne change pas la taille de l'élément, une valeur de 2 double la taille et bien entendu une valeur de 0,5 la divise par deux.

```
transform: scaleX(2);
transform: scaleY(1.5);
transform: scale(2, 1.5);
```

```
transform: scale(2, 0.5);
```

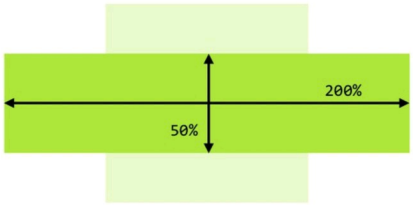


Fig. 8.4 Rendu d'un agrandissement

matrix

Il n'est nullement question ici de pilules ou de lapin blanc, mais d'une fonction mathématique représentant l'ensemble des transformations présentées jusqu'à présent. Il est en effet possible de composer la matrice de transformation combinant l'ensemble des modifications précédentes grâce aux six paramètres de la fonction matrix() et donc d'obtenir la transformation résultante.

En mathématique, une transformation affine 2D peut se rapporter à une matrice de transformation 3×3 dont seules six valeurs seront significatives :

Dans notre cas, cette matrice et ses valeurs a, b, c, d, e, f sont issues de la composition (multiplication) des matrices correspondantes à chacune des transformations vues précédemment. Voici donc les matrices de chaque transformation :

- translation: tx, ty
- échelle: ex, ey

- rotation: α
- inclinaison horizontale: α
- inclinaison verticale: α

Et voici la composition de ces cinq matrices en une seule.

On peut donc récupérer les six valeurs issues de ce calcul et appliquer la règle directement, comme ceci :

```
transform: matrix( 0.725, 0.494 , 0 , 1.655 , 150 , -10);
```

Cette notation est donc équivalente à :

```
transform: translate(150px,-10px) rotate(25deg) skewX(25deg) skewY(10deg) scale( 0.8, 1.5);
```

Comme nous venons de le voir avec cet exemple, il est possible d'appliquer plusieurs transformations à un même élément. Dans ce cas, il suffit de les écrire les unes après les autres séparées par un espace (figure 8.5).

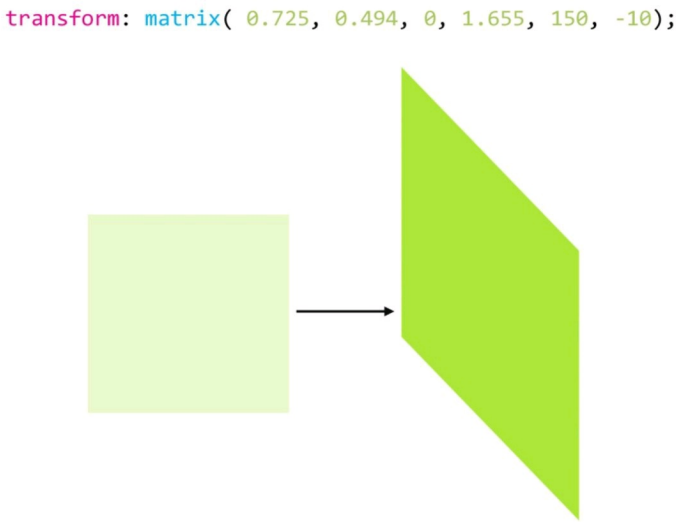


Fig. 8.5 Application d'une matrice de transformation

none

Aucune transformation n'est appliquée. Cette valeur permet aussi d'annuler une ou plusieurs transformations (cas des pseudo-classes ou des *media-queries*).

```
transform: none;
```

8.1.2. transform-origin

Dans tous les exemples vus précédemment, la transformation avait pour origine le centre de l'élément transformé. Il est possible grâce à la propriété transform-origin de choisir un autre point de référence pour appliquer la transformation (figure 8.6). Les valeurs acceptées pour cette propriété sont :

- un pourcentage
- une longueur
- left, center, right pour la valeur horizontale

- top, center, bottom pour la valeur verticale

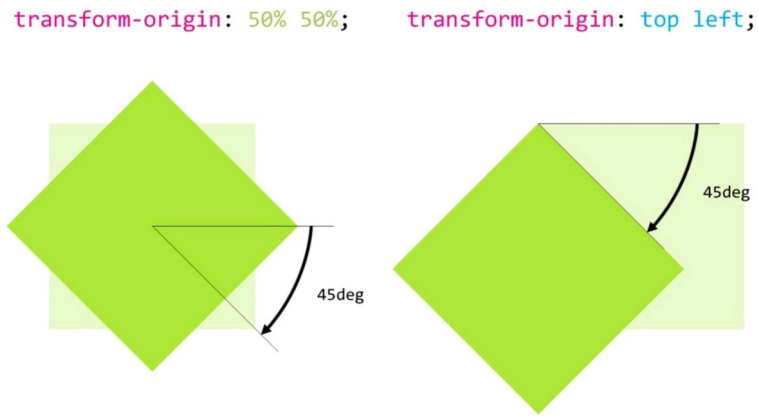


Fig. 8.6 Différence pour l'application d'une rotation à partir du coin supérieur ou du centre.

Transformations 2D	9.0 -ms-	3.5 -moz-	4.0 -webkit-	3.1 -webkit-	10.5 -o-	3.2 -webkit-	11.0 -o-	2.1 -webkit-

Fig. 8.7 Tableau de compatibilité des transformations 2D

8.1.3. À vous de jouer !

Pour illustrer cette partie, nous allons mettre en œuvre la petite transformation du site de démonstration au niveau des caractéristiques de *BugDroid*.

Les transformations sont à voir comme une amélioration progressive. Pour ne pas interférer avec le design initial, nous allons donc réaliser l'insertion du graphique avec la pseudo-classe ::before + content et lui appliquer la transformation. Les navigateurs non compatibles représenteront simplement l'image sans transformation.

Le HTML n'est ici qu'une liste de définition dl et voici le CSS :

```
dl::before{
	content: '';
	display: block;
	width: 80px; height: 80px;
	background: url("mini_droid.svg");
	background-size: 100%;
	position: absolute;
	margin-top: -15px; margin-left: 10px;
	transform-origin: 0 100%;
	transform: rotate(-12deg);
}
```

Quelques explications sont tout de même nécessaires sur ce code. Le format de l'image d'arrière-plan est SVG, il ne faut donc pas oublier de mettre l'image à la bonne taille en précisant background-size:100%. L'image est ensuite mise en position absolue et ajustée avec les propriétés margin-top et margin-left. On applique enfin (figure 8.8) la rotation de –12° à partir du coin en bas à gauche (0 100 %).

Fig. 8.8 Résultat sur le site support

8.2. Les transformations 3D

Nous venons de voir qu'il était possible de transformer un élément quelconque du DOM tout en restant dans le plan représentant la page. En jouant avec les valeurs rotate() et skew(), nous pouvons alors simuler la 3D (figure 8.9).

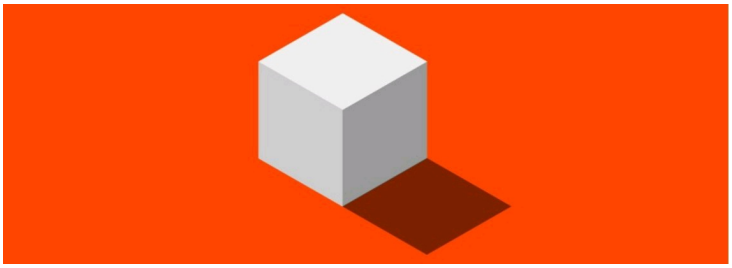


Fig. 8.9 Un iso-cube réalisé avec les transformations 2D

Code CSS de l'iso-cube

```
.face{
	position: absolute;
}
.haut{
	top: 0;
	left: 89px;
	background: #eee;
	transform: rotate(-45deg) skewX(15deg) skewY(15deg);
}
.gauche{
	top: 155px;
	left: 0;
	background: #ccc;
	transform: rotate(15deg) skewX(15deg) skewY(15deg);
}
.droite{
	top: 155px;
	left: 178px;
	background: #999;
	transform: rotate(-15deg) skewX(-15deg) skewY(-15deg);
}
```

Ce rendu, bien que d'aspect graphique satisfaisant n'en reste pas moins de la 2D, et ce n'est pas suffisant pour obtenir des effets 3D digne de ce nom notamment si vous désirez les animer.

La propriété transform a donc été étendue à la troisième dimension afin de transformer les éléments sur

les trois axes, le nouvel axe *z* étant dirigé vers l'utilisateur. De nouvelles propriétés ont aussi été ajoutées pour régler les problèmes liés à la projection sur le plan d'un espace tridimensionnel. Il faut néanmoins garder à l'esprit que ces améliorations sont faites pour améliorer l'expérience utilisateur et, excepté dans le cas d'une application web spécifique, vous devrez donc veuillez à mettre en place une solution alternative cohérente.

Dans un premier temps, nous allons voir les règles spécifiques à la 3D, puis nous poursuivrons avec un exemple concret en place sur le site démo. Voyons tout d'abord les ajouts pour la propriété transform:.

8.2.1. transform

- **Translation**
  - translateZ(z) : translation sur l'axe *z*, les valeurs négatives éloignent l'élément et les positives le rapprochent.
  - translate3d(x,y,z) : translation suivant le vecteur désigné par *x*,*y* et *z*.
- **Rotation**
  - rotateX(x), rotateY(y), rotateZ(z) : rotations autour de l'axe spécifié dans le sens horaire.
  - rotate3d(x,y,z,angle) : rotation autour de l'axe défini par le vecteur (x,y,z) ramené à un vecteur unitaire.
- **Inclinaison**
  - Il n'y a pas d'inclinaison 3D
- **Échelle**
  - scaleX(x) scaleY(y) scaleZ(z) : appliquent un facteur d'échelle sur l'axe spécifié.
  - scale3d(x,y,z) : applique un facteur d'échelle selon les trois axes.
- **Perspective**
  - perspective(n) : spécifie la perspective en px (voir propriété ci après).
- **Complet**
  - matrix3d(a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p) : applique les coefficients de la matrice de transformation 4×4.

Attention, les lettres des paramètres ne correspondent pas à la matrice 2D. Le but de cet ouvrage n'étant pas de faire un cours de mathématiques, nous ne rentrerons pas dans le détail de l'application de cette fonction qui est de toute façon similaire à la fonction analogue pour la 2D.

URL : http://fr.wikipedia.org/wiki/Coordonn%C3%A9es_homog%C3%A8nes

8.2.2. transform-style

La propriété transform-style porte assez mal son nom. Elle est cependant indispensable pour pouvoir représenter une scène 3D composée de plusieurs éléments imbriqués. Elle permet d'indiquer si une transformation 3D est propagée aux enfants de l'arbre DOM. Deux valeurs sont possibles pour cette propriété, flat (à plat) qui est la valeur par défaut et preserve-3d, qui permet cette propagation.

Voici une démonstration pour bien comprendre ce concept. Nous utilisons deux boîtes imbriquées. La première subit une rotation de 45° sur l'axe *y*. La seconde subit un déplacement sur l'axe *z* de 50 px,

suivi d'une rotation sur l'axe *x* de 45°.
Voici le code HTML utilisé :

```
<section>
	<header>
		<h3>BugDroid</h3>
	</header>
</section>
```

Puis le code CSS incluant la perspective sur l'élément section ainsi que les deux transformations 3D des éléments imbriqués. Nous ne spécifions pas encore la valeur de la propriété transform-style qui prend donc la valeur flat par défaut.

```
section{
	perspective: 600px;
}
header{
	width: 150px;
	height: 150px;
	margin: auto;
	background: orangered;
	border: 3px solid darkred;
	transform: rotateY(45deg);
}
h3{
	width: 75px;
	height: 75px;
	background: deepskyblue;
	border: 1px solid dodgerblue;
	margin: 30px auto;
	transform: translateZ(50px) rotateX(45deg);
}
```

Utilisons maintenant la valeur preserve-3d :

```
header{
	...
	transform: rotateY(45deg);
	transform-style: preserve-3d;
}
```

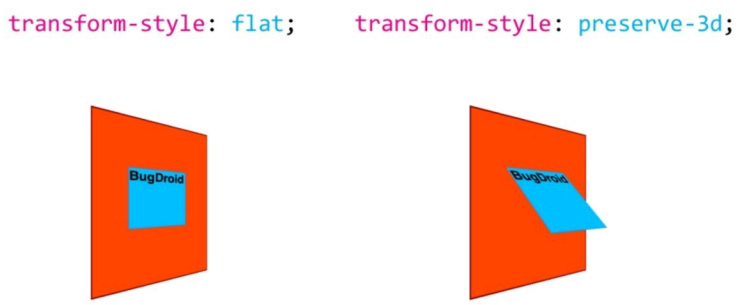


Fig. 8.10 Application de transform-style

Et voici donc le résultat de cette transformation 3D (figure 8.10). Sans preserve-3d, ce n'est pas concluant. L'élément header est correctement transformé mais l'élément h3 n'est pas dans la position attendue. En revanche, on voit qu'il a tout de même été transformé, la rotation sur l'axe des *x* dans le

repère de l'élément header provoquant un aplatissement de l'objet. C'est pourquoi la valeur de cette propriété est nommée flat, la transformation de l'objet a bien lieu, mais son affichage est rendu « à plat » sur son parent. Avec preserve-3d, le rendu est conforme à nos attentes.

Note : la propriété transform-style n'est pas héritée, il faut la réécrire à tous les niveaux de votre scène 3D subissant une transformation.

8.2.3. perspective

La propriété perspective permet de définir la perspective de la vue 3D. La définition de cette propriété est indispensable pour obtenir un rendu 3D. Une vue 3D peut se schématiser par une pyramide dont le sommet est situé au niveau de l'œil de l'utilisateur et dont les arêtes intersectent les coins de la zone visible (le *viewport*). L'application de la propriété perspective permet donc de définir la distance schématique entre utilisateur et la scène 3D. Plus cette distance est grande, plus l'utilisateur est éloigné de la scène et donc moins l'effet 3D est important. À l'inverse, une valeur plus petite rapproche l'utilisateur de la scène. Avec des valeurs trop petites (~< 100 px), certains navigateurs ne font plus le rendu des éléments. Une valeur de 500 px semble être un bon début pour se faire une idée des effets 3D engendrés par les transformations.

Voici un ensemble de schémas qui vont nous éclairer sur l'implication de la perspective sur le rendu de la scène 3D (figure 8.11).

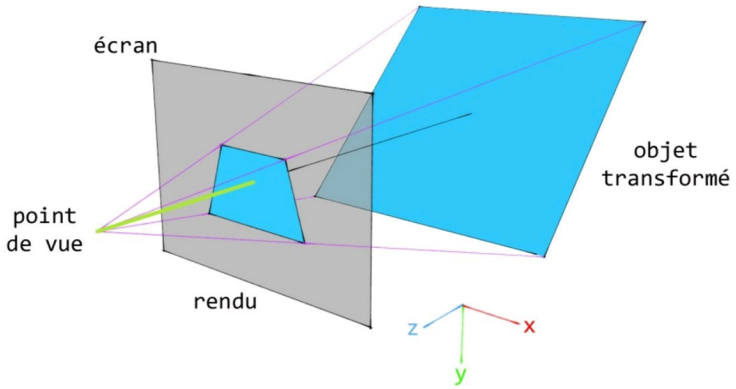


Fig. 8.11 Exemple : la perspective est fixée à 1 000 px, on observe l'effet 3D d'un élément transformé projeté sur l'écran.

La valeur de la perspective, bien qu'abstraite, peut se calculer en fonction de l'effet recherché. Pour le comprendre, reprenons un nouvel exemple (figure 8.12) plus simple avec un élément qui subit une translation de –1 000 px selon l'axe *z*.

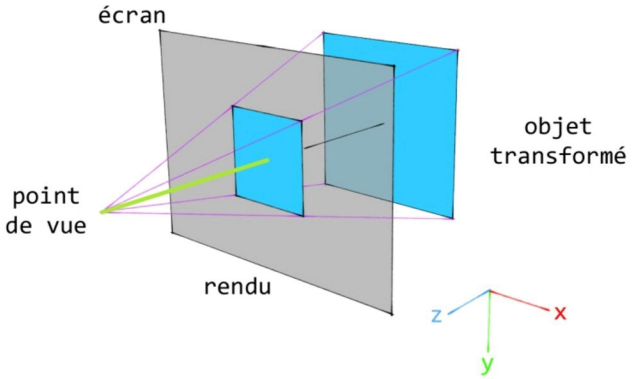


Fig. 8.12 Un objet subit une translation de –1 000 px sur l'axe z – Vue 3D

Dans ce même schéma vu du dessus (figure 8.13), on remarque que l'élément de 1 000 px de large mesure à l'écran exactement 500 px avec une valeur de perspective de 1 000 px.

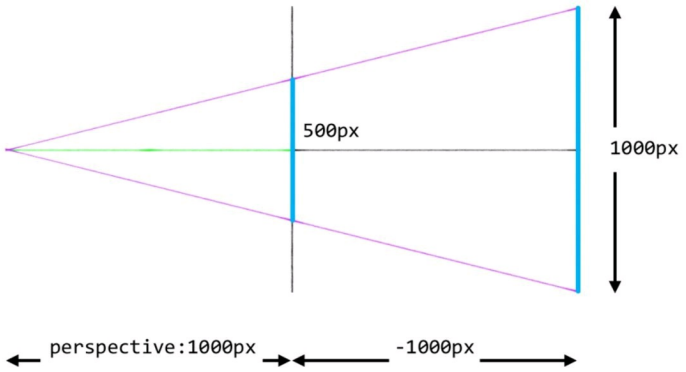


Fig. 8.13 Vue du dessus

Si on ramène la perspective à 500 px, on voit que le cône de projection de l'élément sur l'écran diminue proportionnellement (figure 8.14). L'objet mesure alors exactement 1/3 de 1 000 px soit ~333 px.

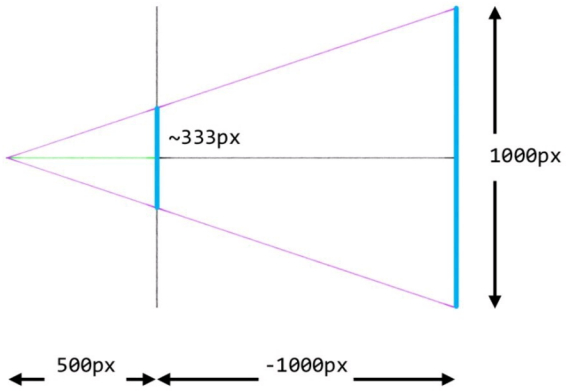


Fig. 8.14 La perspective a été ramenée à 500 px

La propriété perspective peut s'appliquer à un objet parent, appelé couramment « scène ». Les objets enfants subissent cette perspective. Elle peut aussi être précisée comme une valeur de la propriété transform grâce à la fonction perspective(). Cette syntaxe un peu particulière laisse la possibilité de définir une perspective différente pour des objets de la même scène. C'est une possibilité très risquée car les lignes de fuite des objets ne seront plus les mêmes et la scène perdrait alors toute cohérence graphique.

Attention, si vous utilisez les deux techniques, les deux valeurs vont se cumuler. Par exemple :

```
ul{
	perspective: 600px;
}
ul li {
	transform: perspective(600px) rotationX(45deg);
}
```

Équivaut à :

```
ul{
	perspective: 300px;
}
```

```
ul li {
	transform: rotationX(45deg);
}
```

8.2.4. perspective-origin

La propriété perspective-origin permet de définir le centre de projection de la perspective, c'est-à-dire l'endroit où se rejoignent les lignes de fuites de la scène 3D. La valeur par défaut est 50 % 50 %, c'est-à-dire le milieu de l'élément sur lequel elle est appliquée. Il est possible de définir l'origine soit avec les distances, soit avec des mots-clés (top, bottom…), soit en pourcentage par rapport à la dimension de l'objet scène.

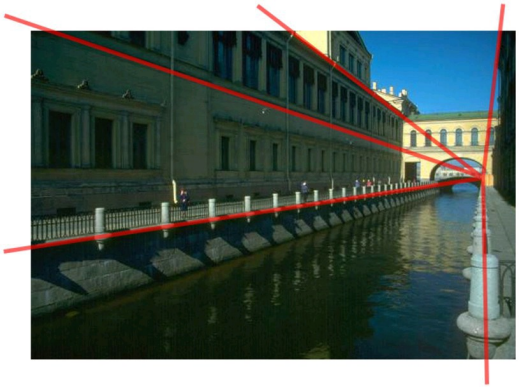


Fig. 8.15 Mise en évidence des lignes de fuite

Les lignes de fuite d'une photographie se coupent en un même point qui correspond à l'origine de la perspective (figure 8.15). C'est ce point que défini la propriété perspective-origin.

Voici deux exemples d'un même objet (une div) ayant subi la même transformation (une rotation de 90° sur l'axe vertical). Nous avons simplement modifié la position du point de jonction des lignes de fuites avec la propriété perspective-origin. Le code HTML et CSS utilisé pour les exemples est le suivant :

HTML

```
<div id="scene">
	<div class="obj"> </div>
</div>
```

CSS

```
#scene{
	background: deepskyblue;
	width: 800px;
	height: 200px;
	perspective: 600px;
	perspective-origin: 50% 50%;
}
.obj{
	height: 200px;
	width: 200px;
	margin: auto;
	background: crimson;
	transform: rotateY(90deg);
}
```

Nous avons donc une scène de dimension fixe, ainsi qu'un objet centré dans la scène par les marges automatiques. Lors de la rotation, l'objet n'est plus visible car il se trouve sur sa tranche.

C'est pourquoi, pour des raisons évidentes de compréhension, nous représentons l'état initial où la valeur par défaut de perspective-origin est égale à 50 % 50 % par un trait (figure 8.16).

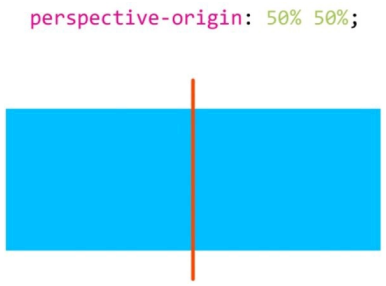


Fig. 8.16 L'élément est sur sa tranche

Essayons avec la valeur suivante :

```
perspective-origin: 0% 50%;
```

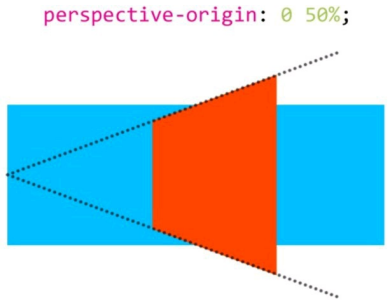


Fig. 8.17 Les points de fuite de l'élément se croisent à l'origine de la perspective

On remarque bien que les lignes de fuites se rejoignent au point correspondant aux valeurs 0 % 50 % (figure 8.17). Le 50 % correspond à la moitié de la hauteur de la div#scene.

Il faut donc considérer la propriété perspective-origin comme un déplacement de la caméra dans le plan d'observation et donc l'utiliser avec précaution.

8.2.5. backface-visibility

La propriété backface-visibility dont le nom signifie « visibilité de la face arrière » permet de spécifier si le navigateur doit représenter les faces arrières des objets subissants une transformation 3D (figure 8.18). Par défaut, la valeur est visible et donc les deux faces de l'objet 3D seront représentées (pas en même temps bien entendu !).

Reprenons l'exemple de la propriété transform-style et appliquons cette fois-ci une rotation de 130° à l'élément header. La face arrière de cet élément est alors visible, masquant la grande partie de l'élément h3[1]. Ajoutons alors la propriété suivante à l'élément header :

```
header{
	...
	backface-visibility: hidden;
}
```

```
backface-visibility: visible;     backface-visibility: hidden;
```

Fig. 8.18 Résultat de l'application de backface-visibility

Cette propriété peut s'avérer très utile pour obtenir un effet de retournement d'objet de type *flipcard*. Les deux faces de l'objet étant des éléments différents, il faut donc rendre invisible l'arrière de chacun d'eux pour ne pas interférer avec l'autre.

Transformations 3D	10.0 -ms-	10.0 -moz-	12.0 -webkit-	4.0 -webkit-		3.2 -webkit-		3.0 -webkit-

Fig. 8.19 Tableau de compatibilité des transformations 3D

Les capacités des navigateurs en matière de transformation 3D dépendent beaucoup de la carte graphique de la machine utilisée car l'accélération est matérielle. Pour cette raison, les éditeurs ont décidé de blacklister certaines cartes graphiques. Le navigateur indique donc qu'il est compatible avec les transformations 3D mais il n'en est rien, le résultat obtenu est alors totalement aléatoire et non-prévisible. Rappelons-nous que tout ceci n'est encore qu'expérimental. Notons également qu'un bug est présent dans l'implémentation de gecko, les paramètres de la propriété perspective-origin sont relatifs au parent et non à l'objet lui-même. Ceci n'est peut-être qu'un bug mais toute autre valeur différente de 50 % 50 % produira un effet différent des autres navigateurs. Nous pouvons aussi évoquer qu'IE10 ne prend pas en charge la valeur preserve-3d de la propriété transform-style.

8.3. Transformations et animations : l'arme absolue ?

Nous terminons ce chapitre avec une application pas à pas des concepts de transformation et d'animation. Nous vous proposons pour cela de concevoir une partie de l'animation d'intro du site de démonstration dont vous avez peut-être déjà réalisé l'aspect dans la partie effets graphiques (figure 8.20).

Fig. 8.20 Animation d'introduction du site support

Cette petite démonstration, simple, permet la mise en œuvre de la plupart des concepts exposés dans les chapitres précédents. Elle vous donnera une base pour la mise en place de vos propres animations. L'accessibilité étant l'une de nos préoccupations, cette animation est basée sur une

amélioration progressive et l'internaute utilisant un navigateur plus ancien verra simplement l'élément graphique sans son animation.

Tout d'abord, regardons la partie HTML. Sur une base de page HTML5 standard, nous posons une section#intro qui correspond à toute la partie d'en-tête du site.

Dans cette section nous avons ensuite deux div, la div#graphikBloc et une div contenant le titre et le menu de navigation.

Concentrons-nous sur la première div.

```
<section id="intro">
	<div id="graphikBloc"></div>
	<div>
		<h1>BugDroid</h1>
		...
	</div>
</section>
```

Le contenu de la div#graphikBloc est exclusivement graphique, réalisé en CSS. Vous en trouverez la méthode de création dans le chapitre sur les effets graphiques. Pour plus de lisibilité, nous la considérons comme un rectangle vert plein dans la suite de la démo.

Attaquons donc le positionnement général de la partie #intro qui va contenir l'animation. Nous précisons simplement les dimensions de cette boîte.

```
#intro{
	height: 310px;
	width: 100%;
}
```

Ensuite, stylons le bloc graphique avec des dimensions et une couleur d'arrière-plan. Ce bloc est flottant à gauche (figure 8.21) :

```
#graphikBloc{
	float: left;
	width: 50%;
	height: 300px;
	padding: 5px;
	margin: 0 20px;
	background: hsla(110, 100%, 40%, 0.8);
}
```

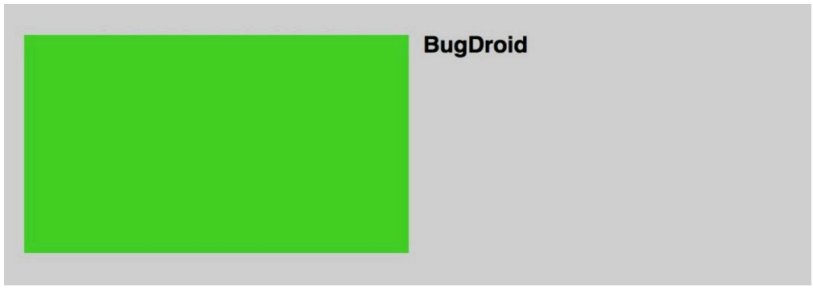


Fig. 8.21 L'élément #graphikBloc est symbolisé en vert

Le décor est posé, il nous reste à faire l'animation.

Commençons par extraire le bloc graphique du champ de vision, en le déplaçant vers la droite et

attribuons l'animation fromright à notre élément.

Les options utilisées sont les suivantes :

- nom de l'animation : fromright ;
- durée de l'animation : 4 secondes ;
- nombre d'itérations : 1 ;
- fonction : cubic-bézier avec accélération finale ;
- état final : forwards, c'est-à-dire position finale.

```
#graphikBloc{
	...
	transform: translateX(1500px);
	opacity: 0;
	animation: fromright 1.8s 1 cubic-bezier(1, 0.2, 0.8, 0.5) forwards;
}
```

La première version de l'animation fromright :

```
@keyframes fromright {
	from{
		opacity: 0;
		transform: translateX(1500px);
	}
	to{
		opacity: 1;
		transform: translateX(0);
	}
}
```

Essayez le rendu de cette animation. Si tout est correct, le bloc graphique doit arriver tranquillement de la droite avec un effet de fondu. Un peu trop tranquillement à votre goût ? C'est normal, nous avons précisé 1,8 seconde pour l'animation, ce qui est la temporisation totale.

Avant de réaliser la seconde partie de l'animation, ajoutons deux étapes à 50 % et à 65 % et précisons la valeur transform: translateX(0). Ainsi l'animation dure toujours 1,8 seconde, mais le déplacement ne dure que 0,9 seconde. Vous comprendrez l'utilité de l'étape à 65 % juste après.

```
@keyframes fromright {
	from{
		opacity: 0;
		transform: translateX(1500px);
	}
	50%{
		opacity: 1;
		transform: translateX(0);
	}
	65%{
		transform: translateX(0);
	}
	to{
		opacity: 1;
		transform: translateX(0);
	}
}
```

Pour réaliser la transformation 3D il nous faut définir plusieurs choses :

- la perspective ;
- l'origine de la transformation.

Premièrement, ajoutons la perspective à l'objet #intro.

```
#intro {
	...
	perspective: 600px;
}
```

Précisons l'origine des transformations de l'élément #graphikBlock : en haut au centre.

```
#graphikBloc{
	...
	transform-origin: 50% 0%;
}
```

Effectuons ensuite la transformation 3D, une rotation de 12° selon l'axe des *y* entre les temps 65 % et 100 %, ce qui crée une pause entre le temps 50 % et 65 % :

```
@keyframes fromright {
	...
	65%{
		transform: translateX(0) rotateY(0);
	}
	to{
		opacity: 1;
		transform: translateX(0) rotateY(12deg);
	}
}
```

Nous ajoutons également une fonction d'accélération différente pour cette transformation :

```
@keyframes fromright {
	...
	65%{
		transform: translateX(0) rotateY(0);
	    animation-timing-function: ease-out;
	}
	to{
		opacity: 1;
	transform: translateX(0) rotateY(12deg);
	}
}
```

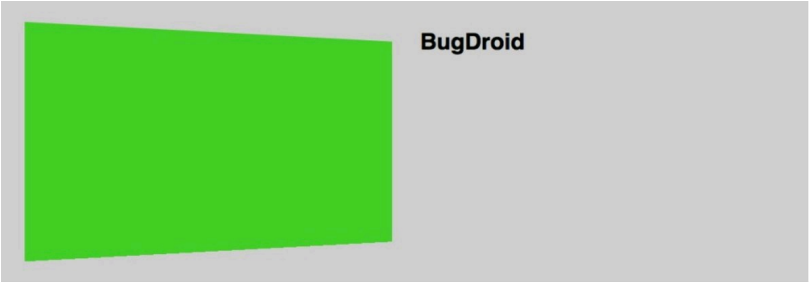


Fig. 8.22 L'élément #graphikBloc a fini son animation

Nous venons donc de réaliser une animation avec quelques lignes de CSS3 (figure 8.22). Mais avons-nous réellement fait un développement en amélioration progressive ? A priori non ! Pour réaliser notre effet sans interagir avec les navigateurs non compatibles, nous allons utiliser la bibliothèque Modernizr dont le principe est détaillé dans le chapitre sur les outils et dont voici le concept : la balise html se voit attribuer une classe correspondant à une fonctionnalité si, et seulement si, cette fonctionnalité est supportée par le navigateur de l'utilisateur. Il n'est pas obligatoire d'utiliser cette technique dans le cadre de votre apprentissage et vous pouvez donc omettre les classes en questions sans que cela ne nuise à la réalisation de l'effet.

Pour notre exemple, nous testons donc ici la présence de la classe .cssanimations pour l'animation de notre bloc graphique. Le test de la classe .csstransforms3d pour la transformation 3D est inutile car un navigateur compatible avec les animations et incompatible avec les transformations 3D ne réalisera pas cette dernière sans autre conséquence.

Notre code final devient donc simplement (sans préfixes) :

```
#intro{
	height: 430px;
	width: 100%;
}
.csstransform3d #intro{
	perspective: 600px;
}
#graphikBloc{
	float: left;
	width: 50%;
	height: 300px;
	padding: 5px;
	margin: 20px;
	background: hsla(110,100%,40%,.8);
}
.cssanimations #graphikBloc{
	opacity: 0;
	transform: translateX(1500px);
	transform-origin: 50% 0%;
	animation: fromright 1.8s 1 cubic-bezier(1, 0.2, 0.8, 0.5) forwards;
}
@keyframes fromright {
	from{
		opacity: 0;
		transform: translateX(1500px);
	}
	50%{
		opacity: 1;
```

```
            transform: translateX(0);
        }
        65%{
            transform: translateX(0) rotateY(0);
            animation-timing-function: ease-out;
        }
        to{
            opacity: 1;
            transform: translateX(0) rotateY(12deg);
        }
}
```

Ce qu'il faut retenir...

Avec les transformations, CSS3 propose un nouvel outil vraiment performant dont les arcanes n'ont pas encore été découverts. Étant donné la bonne prise en charge dans les navigateurs modernes, IE10 compris, elles forment avec les animations un duo qui devrait sans surprise s'imposer dans le domaine des animations de pages web. La concurrence est pourtant bien là. Face à canvas d'HTML5, WebGL ou encore Flash, le duo à un avantage, celui d'être totalement intégré à la mise en forme. En sa défaveur cependant, sa prise en main n'est pas des plus aisées. Cette technologie réussira à s'imposer lorsque de vrais outils de créations d'animations seront disponibles, à l'image de l'IDE[2] de Flash. Notons qu'il existe déjà des applications qui tentent de réaliser cela, citons entre autre Sencha Animator et Adobe Edge. C'est également d'outils, mais d'un autre ordre, que nous allons vous parler dans le chapitre qui suit.

Notes

[1] Sur les navigateurs ayant une bonne implémentation du moteur 3D (FF12 – Chrome 17).

[2] Environnement de développement intégré.

Outils et ressources CSS

Objectifs

Afin d'offrir une expérience similaire entre les différents navigateurs, de nombreuses solutions peuvent êtres mises en place. Comme nous l'avons vu tout au long de cet ouvrage, certaines sont simples, d'autres sont plus difficiles et il est même parfois impossible d'obtenir un rendu identique partout.

L'objectif de ce chapitre est de vous présenter les différents outils et ressources utiles pour l'intégration web de fonctionnalités CSS3. Certains de ces outils sont indispensables dans la mise en place de la dégradation gracieuse ou de l'amélioration progressive.

9.1. CSS3 PIE : séduisant !

S'il est un outil séduisant, c'est bien celui-ci. En effet, CSS3 PIE (comprenez CSS3 *Progressive Internet Explorer*) permets, comme son nom l'indique, de pouvoir reproduire les effets graphiques de CSS3 dans les navigateurs de IE6 à IE8, voire même IE9, qui n'implémentent pas certaines propriétés nativement. Et cela, très facilement !

Le site que vous venez de faire, tout beau, tout frais, avec plein d'effets en CSS3 ne vous plaît pas sous IE ? C'est normal, puisque la majorité de ces effets ne sont pas reconnus. Grâce à CSS3 PIE, Internet Explorer sera capable de les reproduire, sans aucuns efforts.

Fig. 9.1 Logo de CSS3 PIE

En termes de dégradation gracieuse, CSS3 PIE peut être la solution. En effet, cela nous permet d'obtenir un rendu similaire très rapidement. Attention toutefois, cette solution n'est pas universelle, puisque qu'elle ne fonctionne que sur IE. Il faut donc prévoir un *fallback*[1] pour tous les autres navigateurs incompatibles. Ce *fallback* est peu être suffisant et donc CSS3 PIE n'est pas toujours obligatoire...

9.1.1. Quels effets graphiques ?

CSS3 PIE rends les versions 6 à 8 d'Internet Explorer capable d'interpréter les propriétés décoratives CSS3 suivantes :

- border-radius
- box-shadow
- border-image
- images d'arrières-plans multiples

- dégradés linéaires

Note : pour IE 9, seuls border-image et les dégradés linéaires (linear-gradient) sont ajoutés, car les autres propriétés sont déjà nativement interprétées. Aucune de ces propriétés n'est ajoutée à IE10, même si celui-ci ne reconnaît pas encore border-image.

9.1.2. Comment ça marche ?

Le principe est d'ajouter en CSS un comportement spécial aux éléments qui utilisent des règles décoratives CSS3. L'ajout de ce comportement passe par l'utilisation d'un fichier HTC[2] et de la propriété CSS propriétaire behavior. Dans CSS3 PIE, ce fichier s'appelle PIE.htc.

Pour utiliser CSS3 PIE, téléchargez tout d'abord le ZIP contenant les fichiers nécessaires sur le site officiel (http://css3pie.com) et décompressez l'archive. À l'intérieur, vous retrouvez PIE.htc : c'est le seul fichier dont nous avons besoin pour le moment. Déposez-le sur votre site, puis ajoutez le comportement sur un élément HTML qui utilise des propriétés décoratives, *via* une règle CSS :

Ajout du comportement à un élement dont class="border-radius"

```
.border-radius{
      -webkit-border-radius: 20px;
         -moz-border-radius: 20px;
              border-radius: 20px;
      behavior: url('chemin/PIE.htc');
}
```

Ainsi, l'élément qui reçoit ce comportement est modifié pour les différentes versions d'Internet Explorer, et en l'occurrence, la boîte se retrouve avec des coins arrondis (border-radius). Visualisez le rendu sur Internet Explorer (figure 9.2) ! C'est magique...

Fig. 9.2 Exemple de simulation de styles avec CSS3 PIE

Attention, le chemin du fichier PIE.htc doit être relatif au fichier HTML, et non au fichier CSS comme c'est le cas normalement. Cela peut poser quelques problèmes que nous détaillerons plus loin.

Le principe de fonctionnement de CSS3 PIE est celui-ci: tout d'abord, lorsque le nouveau comportement est appliqué, une balise <css3-container /> est créée. Ce conteneur est inséré en tant que frère précédent l'élément cible, et positionné de manière absolue aux mêmes coordonnées. Cette balise contiendra tous les objets VML[3] nécessaires au rendu visuel demandé. Cet élément crée donc « l'illusion ».

Voici un aperçu de code généré (figure 9.3) : une balise <css3-container> contient trois balises : <outset-

box-shadow>, <background> et <border-image> lesquelles contiennent un objet VML. Ces trois objets recréent visuellement un élément sur lequel est appliqué une ombre portée (box-shadow), un dégradé linéaire (background) et une bordure imagée (border-image).

```
<!-- DOCTYPE html -->
<html>
  <head>
  <body>
    <div>
      <css3-container style="z-index: auto; position: absolute; direction: ltr; top: 200px; left: 208px;">
        <outset-box-shadow style="position: absolute; top: 0px; left: 0px;">
          <group1>
            <css3vml:shape style="position: absolute; width: 354px; height: 254px; top: 0px; behavior: u
              <css3vml:fill/>
              <css3vml:fill/>
        <background style="position: absolute; top: 0px; left: 0px;">
        <border-image style="position: absolute;">
      <p class=" pie_first-child  " _pieId="_15">
```

Fig. 9.3 Un exemple de code généré sous IE8

9.1.3. Limitations

Simuler le rendu visuel des propriétés CSS3 dans des navigateurs qui ne l'implémentent pas n'est pas chose aisée, c'est pourquoi il existe quelques limitations quant à l'utilisation de celles-ci. Voici le détail propriété par propriété.

Coins arrondis (border-radius)

- CSS3 PIE simule border-radius pour IE6 à IE8. IE9 et suivants ont un support natif.
- Seule la syntaxe raccourcie est acceptée (pas de border-top-left-radius ni autres propriétés longues).

Note : privilégiez toujours la syntaxe raccourcie, même pour spécifier un seul angle arrondi, comme ceci border-radius:0 20px 0 0;.

Ombres de boîtes (box-shadow)

- CSS3 PIE simule box-shadow pour IE6 à IE8. IE9 et suivants ont un support natif.
- L'ombre générée n'est pas découpée sous l'élément, comme c'est le cas dans les autres navigateurs, et apparaît donc opaque. Lors de l'utilisation d'un fond transparent ou semi-transparent, l'ombre est alors visible au travers. Elle devrait pourtant être rognée au niveau des bordures.
- Le mot-clé inset n'est pas supporté. Les ombres internes ne peuvent pas être reproduites.

Note : l'utilisation d'un arrière-plan non transparent peut vous aider à régler le premier problème.

Bordures imagées (border-image)

- CSS3 PIE simule border-image pour IE6 à IE9. IE10 n'est pas pris en charge même si cette propriété n'est pas (encore) supportée nativement.
- Le mode de répétition des images est uniquement stretch pour IE6-8. Les modes repeat et round sont supportés pour IE9.
- border-image-outset n'est pas reconnu, mais ne l'est pour aucun autre navigateur actuellement.
- Le mot-clé fill est reconnu, et doit être présent pour fonctionner, mais pour la majorité des

navigateurs, c'est l'inverse. Il est alors difficile de pallier ce problème.
- Le chemin des images est relatif au fichier HTML et non au CSS.

Note : la dégradation gracieuse est fortement encouragée pour les bordures imagées.

Images d'arrières plans multiples

- CSS3 PIE simule les images d'arrière-plans multiples pour IE6 à IE8. IE9 et suivants ont un support natif.
- La propriété background doit être préfixée, comme ceci : -pie-background.
- Le chemin des images est relatif au fichier HTML et non au CSS.

Bien que -pie-background reconnaisse l'utilisation des propriétés d'arrière-plans, IE ne les interprète pas, et donc :

- background-attachment vaut scroll, même si fixed ou local sont précisés.
- background-size ne s'applique pas.
- background-repeat ne reconnaît pas les mots-clés space ou round.
- background-origin vaut toujours padding-box.
- background-position avec plus de deux valeurs ne s'applique pas.

Note : la dégradation gracieuse est fortement encouragée.

Dégradés CSS

- CSS3 PIE simule les dégradés pour IE6 à IE9. IE10 a un support natif.
- Seuls les dégradés linéaires sont utilisables. Les dégradés radiaux ne peuvent pas être reproduits.
- Les dégradés linéaires ne sont utilisables qu'avec la propriété préfixée -pie-background, comme ceci : -pie-background: linear-gradient(#555,#333);.

Les limitations qui suivent sont uniquement valables pour IE6-8 (IE9 reproduit les dégradés fidèlement) :

- Les *color-stops* ne peuvent pas être transparents ou semi-transparents. Si une valeur rgba est utilisée, la valeur rgb sera appliquée.
- Les *color-stops* dont la valeur se situe à l'extérieur de la boîte (e.g. 150 %) ne sont pas supportés.
- Les propriétés background-size/origin/repeat/position ne sont pas supportées.

La dégradation gracieuse sur tous les dégradés complexes est vivement recommandée. De plus, pour un rendu final « identique » sur tous les navigateurs, la syntaxe pour un unique dégradé serait la suivante :

Ajout d'un dégradé pour tous les navigateurs

```
#element {
	background: #CCC; /*fallback pour les vieux navigateurs*/
	background: -webkit-linear-gradient(#CCC, #EEE); /*webkit*/
	background:    -moz-linear-gradient(#CCC, #EEE); /*gecko*/
	background:     -ms-linear-gradient(#CCC, #EEE); /*IE*/
	background:      -o-linear-gradient(#CCC, #EEE); /*opera*/
```

```
    background:         linear-gradient(#CCC, #EEE); /*spec*/
      -pie-background: linear-gradient(#CCC, #EEE); /*PIE*/
    behavior: url(PIE.htc);
}
```

Il convient alors de savoir si votre dégradé est primordial pour votre site, car la maintenance de telles règles n'est pas une chose aisée.

rgba

CSS3 PIE simule la fonction rgba pour IE6 à IE8 ce qui permet d'utiliser la notion de transparence. Cependant, l'utilisation de rgba n'est possible que dans les cas suivants :

- La couleur est spécifiée *via* -pie-background.
- La couleur est celle d'une ombre (box-shadow) si celle-ci n'est pas floue.
- La couleur des *color-stops* pour les dégradés (dans IE9 uniquement).

Dans les autres cas d'utilisation des propriétés décoratives reconnues, la couleur appliquée est celle correspondante, mais *via* la fonction rgb.

9.1.4. Résolutions de bugs

Comme tout outil qui tente de combler un manque de compatibilité, CSS3 PIE comporte encore quelques bugs au niveau de la simulation des différents effets pour Internet Explorer. La plupart de ces bugs ont déjà été identifiés et des solutions, non efficaces à 100 %, ont été trouvées pour tenter de les contourner. Voici un aperçu des principales résolutions de bugs.

Problème avec z-index (disparition des effets)

Étant donné le principe de fonctionnement de CSS3 PIE vu plus haut, la simulation des effets par un élément conteneur VML fonctionne parfaitement lorsque l'élément cible est en position: absolute ou relative, car le conteneur est positionné au même z-index.

Le problème peut se poser lorsque l'élément cible est en position: static (cas normal), car l'élément conteneur ne se voit pas forcément attribuer un bon z-index. Cela peut poser des problèmes de superposition, ou de disparition des effets.

Il faut alors forcer le positionnement, comme ceci :

- soit avec une position: relative sur l'élément cible ;
- soit avec une position: relative sur l'élément parent, et lui appliquer un z-index.

> **URL** : http://css3pie.com/documentation/known-issues/#z-index

Chemins relatifs

Il existe deux limitations concernant les chemins relatifs :

- le chemin utilisé avec la propriété behavior doit être relatif à la page HTML, et non au fichier CSS.
- les URLs utilisées avec border-image, ou avec -pie-background doivent être relatives à la page HTML également.

Il est alors possible d'utiliser des URLs absolues, mais cela peut induire d'autres désagréments.

> **URL** : http://css3pie.com/documentation/known-issues/#relative-paths

Problèmes sur certains types d'éléments

Il n'est pas possible de simuler des effets appliqués sur les éléments body et html avec les versions 7 et inférieures, même des dégradés CSS. IE8 s'en sort très bien.

Évitez aussi l'utilisation de CSS3 PIE sur les balises fieldset, car l'élément legend peut ne pas être affiché correctement.

De plus, les éléments ne contenant aucun fils (comme img ou input) peuvent poser problème si leur taille est spécifiée en em ou en ex.

> **URL** : http://css3pie.com/documentation/known-issues/#elements

Autres

D'autres solutionnements de bugs sont référencés sur le site de CSS3 PIE, notamment :

- le problème de *hasLayout* d'Internet Explorer ;
- la gestion unique de la syntaxe raccourcie des propriétés CSS3 ;
- la limitation de l'appel PIE.htc au même domaine ;
- des problèmes de configuration de serveur ;
- etc.

> **URL** : http://css3pie.com/documentation/known-issues/

9.2. Modernizr : simple et puissant !

Modernizr est une bibliothèque JavaScript très légère qui détecte les implémentations de votre navigateur en ce qui concerne les nouveautés d'HTML5, de CSS3 et d'un soupçon d'autres choses, notamment SVG (figure 9.4).

Fig. 9.4 Logo de Modernizr

La méthode utilisée est très fiable puisque qu'elle ne repose pas sur le principe de *UA sniffing* (lecture du *User-Agent* du navigateur) mais bien sur la détection des fonctionnalités implémentées dans le navigateur du client. Modernizr nous laisse ensuite utiliser ce qu'il a détecté de plusieurs façons :

- avec la création d'un objet JavaScript, nommé Modernizr, où chacun de ses attributs contient le résultat des tests sous forme booléenne ;
- par l'ajout de classes sur la balise html qui précise quelles fonctionnalités sont supportées nativement ou non (figure 9.5).

```
<!DOCTYPE html>
<html class=" js no-flexbox rgba hsla multiplebgs backgroundsize borderimage borderradius boxshadow textshadow
 opacity cssanimations csscolumns cssgradients no-cssreflections csstransforms csstransforms3d csstransitions
 fontface generatedcontent svg inlinesvg smil svgclippaths"><head>
        <title>Bugdroid - Livre CSS3</title>
        <meta charset="utf-8">
        <meta id="viewport" name="viewport" content="width=device-width,initial-scale=1,maximum-scale=10">
```

Fig. 9.5 Ajout automatique de classe sur l'élément html

Nous avons donc la possibilité d'utiliser tels ou tels aspects de la spécification en étant parfaitement sûr que le navigateur du client puisse l'utiliser. Cela nous permet surtout de mettre en place des solutions alternatives pour les navigateurs qui ne supportent pas ces nouveautés, dans le cadre de l'amélioration progressive.

9.2.1. Installation

Il n'existe pas d'installation à proprement parler. Il suffit simplement de faire appel à la librairie. Pour la récupérer, deux techniques sont disponibles :

- télécharger le fichier JavaScript complet, puis l'appeler à l'aide d'une balise script (ou l'appeler depuis le site officiel) ;
- télécharger un fichier sur-mesure afin de gagner encore quelques kilo-octets, puis l'appeler *via* la balise script. Cette possibilité de création sur-mesure est d'une facilité déconcertante, grâce au formulaire dédié (figure 9.6).

Appel de la librairie non compressée depuis le site officiel

```
<script type="text/javascript" src="http://www.modernizr.com/downloads/modernizr-latest.js"></script>
```

Appel de la librairie sur-mesure depuis votre hébergement

```
<script type="text/javascript" src="chemin/modernizr-custom.js"></script>
```

Cet appel doit être fait dans la balise head de votre document HTML, et de préférence après l'appel aux feuilles de styles.

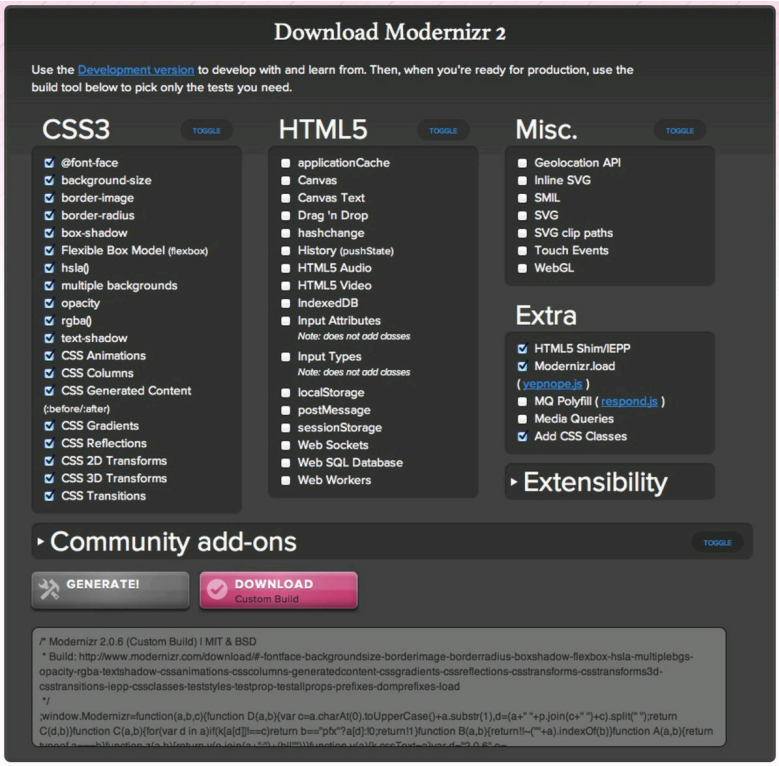


Fig. 9.6 Création de la librairie sur-mesure

9.2.2. Fonctionnalités supportées

Voici une liste non exhaustive des fonctionnalités CSS testées par Modernizr, ainsi que les noms des classes ajoutées à l'élément html. Le nom est identique pour les attributs de l'objet JavaScript Modernizr qui est créé (tableau 9.1).

Tab. 9.1

Fonctionnalités CSS	Classe ajoutée
@font-face	fontface
background-size	backgroundsize
border-image	borderimage
border-radius	borderradius
box-shadow	boxshadow
Flexible Box Layout	flexbox
hsla	hsla
Plusieurs arrières-plans	multiplebgs
opacity	opacity
rgba	rgba
text-shadow	textshadow
Multicolonnes	csscolumns
Pseudo-éléments :after/:before	generatedcontent
Dégradés CSS	cssgradients
Réflections CSS	cssreflections
Transformations2D	csstransforms
Transformations3D	csstransforms3d
Transitions CSS	csstransitions
Animations CSS	cssanimations

Les noms des classes de ce tableau sont préfixées de no- si la fonctionnalité n'est pas reconnue. Par exemple, un navigateur ne prenant pas en charge les dégradés aura une classe no-cssgradients ajoutée au sein de la balise html du document. Il devient donc possible de prévoir une solution alternative pour ces navigateurs en ciblant des éléments via cette classe.

Les fonctionnalités détectées de HTML5 ou de SVG ne sont pas répertoriées dans ce livre. Reportez-vous à la documentation officielle pour les découvrir.

URL : http://www.modernizr.com/docs/

9.2.3. Utilisation basique

Modernizr nous permet de mettre en place une solution alternative pour les navigateurs qui ne supportent pas certaines fonctionnalités CSS3, ou à l'inverse de proposer des solutions qui ajoute une plus-value à nos pages web. Ces solutions passent par le ciblage d'élément *via* CSS en utilisant les classes ajoutées dans la balise html. Prenons par exemple les transformations CSS 3D : celles-ci sont interprétées dans un nombre limité de navigateurs. Les transformations 2D sont elles reconnues plus largement, mais certains navigateurs ne sont pourvus d'aucune de ces fonctionnalités. Les classes csstransforms3d, csstransforms et no-csstransforms3d ajoutées sur l'élément html vont nous êtres utiles. Nous pourrions alors procéder ainsi :

Ciblage d'un élément, peu importe le support du navigateur

```
#element{
	/* style pour tous les navigateurs */
}
```

Ciblage d'un élément, enfant de html qui à la classe .csstransforms3d

```
html.csstransforms3d #element{
	/* styles pour les navigateurs compatibles transform 3D */
}
```

Ciblage d'un élément, enfant de html qui à la classe .csstransforms ET la classe .no-csstransforms3d

```
html.csstransforms.no-csstransforms3d #element{
	/* styles pour les navigateurs compatibles transform 2D, mais pas 3D */
}
```

Voici un autre exemple, l'utilisation des dégradés CSS avec la détection de Modernizr :

Ciblage d'un élément dans un navigateur ne prenant pas en charge les dégradés

```
.no-cssgradients #element{
	/* chargement d'une image d'arrière-plan */
	background-image: url('chemin/image.png');
}
```

Ciblage d'un élément dans un navigateur prenant en charge les dégradés

```
.cssgradients #element{
	/* création du dégradé en CSS ( sans oublier les versions préfixées) */
	background-image: linear-gradient(left, #555, #333);
}
```

Ou encore l'ajout d'une bordure en lieu et place d'une ombre :

Ciblage de tous les éléments

```
#element{
	box-shadow: 0 0 5px black;
}
```

Ciblage des éléments ne prenant pas en charge box-shadow

```
.no-boxshadow #element{
	border: 1px solid black;
}
```

9.2.4. Utilisation avancée

Une utilisation poussée de Modernizr nous permet d'aller encore plus loin avec les CSS (et les autres fonctionnalités), comme par exemple l'interaction avec JavaScript. En effet, il est possible d'ajouter ses propres tests afin de détecter des fonctionnalités CSS non testées par défaut. Modernizr peut également nous aider à créer notre JavaScript en fonction du support du navigateur ou à préfixer les fonctionnalités non encore standard, comme les nouveaux évènements de transition ou d'animation CSS. La documentation de Modernizr vous apportera tous les détails nécessaires à son utilisation.

9.2.5. Limitations

La principale limitation quant à l'utilisation de Modernizr est le fait que la solution repose entièrement sur JavaScript. Si l'utilisateur a désactivé JavaScript, aucun style n'est appliqué, puisque les classes ne sont pas ajoutées sur l'élément html. Il convient donc de prévoir des styles par défaut, sans ciblage de l'élément parent, comme dans les exemples ci-dessus.

9.3. Selectivizr

Selectivizr est une bibliothèque JavaScript qui permet d'émuler les sélecteurs CSS3 inconnus par IE6-8. Cela permet notamment de pouvoir utiliser les nouveaux sélecteurs d'attributs et de pseudo-classes dans ces navigateurs, sans modification de notre code.

Fig. 9.7 Logo de Selectivizr

9.3.1. Installation

Comme pour Modernizr, il n'y a pas d'installation à faire puisque le principe de fonctionnement repose sur une librairie JavaScript. Il suffit simplement de faire appel à cette librairie depuis l'entête de vos pages web. Cependant, cette librairie nécessite un *framework* JavaScript pour fonctionner. Rassurez-vous, les *frameworks* compatibles sont renommés, et vous utilisez peut être déjà l'un d'entre eux sur vos sites. Il s'agit de jQuery, Mootools, Prototype, dojo ou encore YUI.

Pour que Selectivizr fonctionne, il faut alors faire l'appel à l'un des *frameworks* ci-dessus, avant l'appel à Selectivizr.

Appel du framework, puis de la librairie

```
<script type="text/javascript" src="framework.js"></script>
...
<script type="text/javascript" src="selectivizr.js"></script>
```

Par souci de performance, l'appel à Selectivizr devrait être placé au sein de commentaires conditionnels afin de ne cibler que les versions d'Internet Explorer nécessaires. Par exemple, les versions d'IE9 et suivantes ainsi que les versions de Windows Phone ont un très bon support des sélecteurs CSS, il est donc inutile d'ajouter Selectivizr à ces versions. Voici un exemple d'appel conditionnel concernant les versions d'IE inférieure ou égale à 8 et n'incluant pas les versions

mobiles :

Appel de la librairie en fonction du navigateur

```
<!–[if (lte IE 8) & (!IEMobile)]>
      <script type="text/javascript" src="selectivizr.js"></script>
      <noscript><link rel="stylesheet" href="alternative.css" /></noscript>
<![endif]–>
```

L'ajout d'une balise noscript permet de gérer le cas ou le client utilise un navigateur dont JavaScript est désactivé. Cela nous permet de prévoir une feuille de styles alternative dans ce cas précis.

9.3.2. Fonctionnalités supportées

Selectivizr ajoute le support des sélecteurs CSS3 (et CSS2.1) suivants :

[attribut], [attribut=valeur], [attribut~=valeur], [attribut|=valeur], [attribut□=valeur], [attribut$=valeur], [attribut*=valeur], :nth-child, :nth-last-child, :nth-of-type, :nth-last-of-type, :first-child, :last-child, :only-child, :empty, :enabled, :disabled, :checked, :hover, :focus, :target, :not, :root, ::first-line, ::first-letter

Cette liste détaillée, avec le support par framework, est disponible sur le site officiel.
URL : http://www.selectivizr.com

9.3.3. Utilisation

L'utilisation est on ne peut plus simple, la librairie se charge de tout. En effet, une fois celle-ci chargée, les sélecteurs CSS présents dans vos feuilles de styles sont instantanément interprétés par les versions d'Internet Explorer ciblées. Les styles sont alors directement appliqués. Aucune modification n'est nécessaire, et ça c'est bien.

9.3.4. Limitations

Il existe plusieurs limitations avec l'utilisation de Selectivizr :

- si JavaScript est désactivé chez l'utilisateur, les styles utilisant des sélecteurs CSS3 ne seront pas appliqués ;
- la pseudo-classe :target n'est supportée qu'avec IE8 ;
- les styles étant appliqués une seule fois au chargement de la page, toute modification du DOM ne sera pas reflétée.
- les feuilles de styles doivent êtres ajoutées via l'élément link ou via @import. Seule la balise style n'est pas parsée.
- les feuilles de styles doivent êtres hébergées sur le même domaine que la page appelante.
- tous les sélecteurs ne sont pas simulables. Cela dépend du *framework* choisi (figure 9.8).

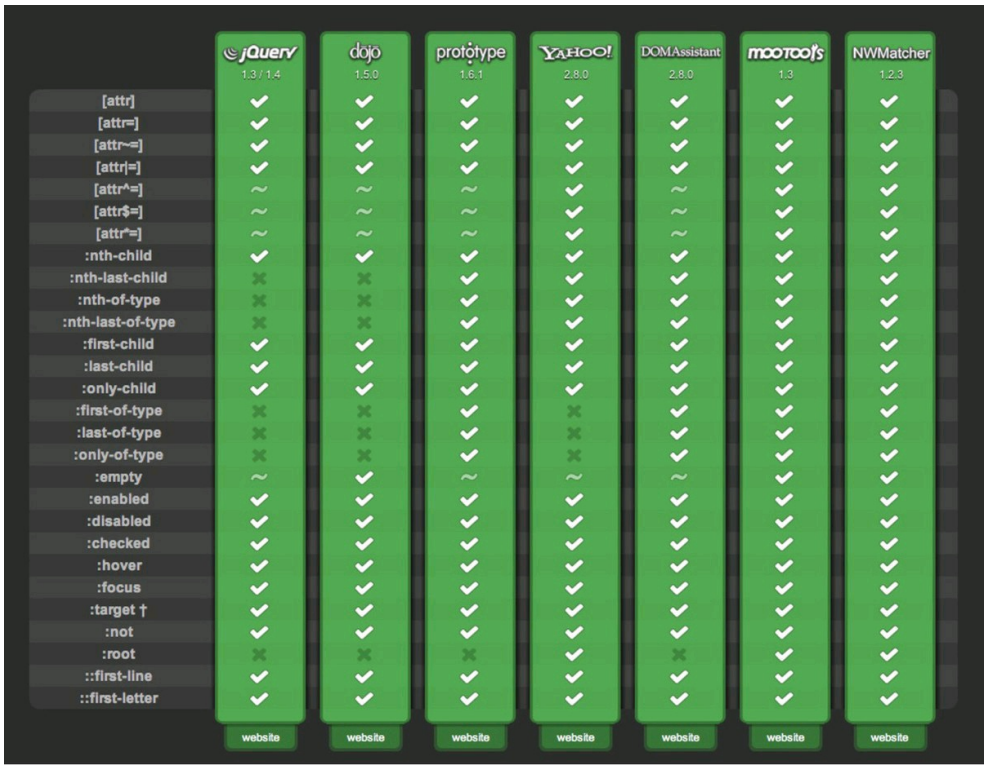


Fig. 9.8 Sélecteurs CSS3 simulables en fonction du framework

9.4. Préfixeurs CSS

Pour utiliser CSS3 dès aujourd'hui, nous devons sans cesse préfixer nos propriétés CSS pour que celles-ci soient interprétées dans les différents navigateurs du marché. Ces préfixes sont nécessaires tant que la spécification définie par le W3C n'a pas atteint le stade de recommandation. Nous nous devons d'écrire plusieurs fois la même règle CSS, enrichie des préfixes navigateurs, pour un support optimal de notre mise en page. Dans les faits, cela revient souvent à écrire trois, quatre voire cinq fois la même chose, comme nous vous l'avons conseillé tout au long de cet ouvrage (figure 9.9)

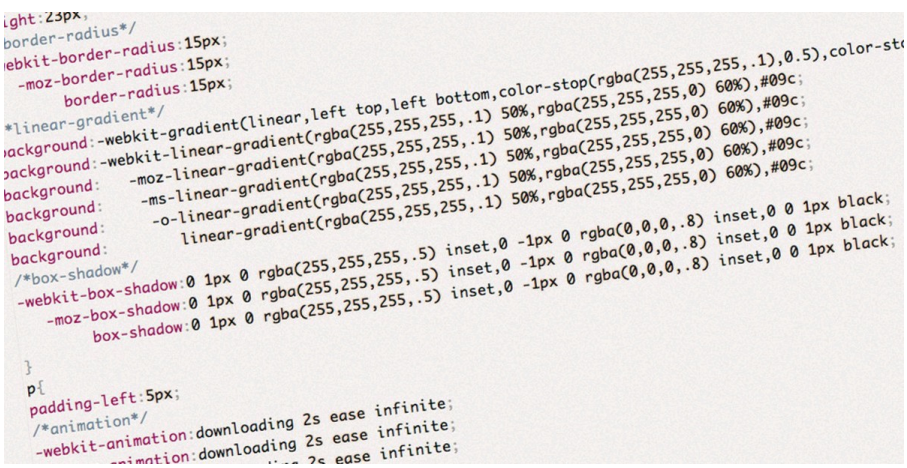


Fig. 9.9 Code avec préfixes pour un support complet

Le problème est encore plus accentué lorsque les spécifications sont modifiées alors même que certains navigateurs en offraient déjà un support. Mais c'est le principe même de l'utilisation des préfixes. C'est le cas notamment des dégradés CSS ou du modèle de boite *Flexbox*.

La mise à jour d'un site est également rendue plus contraignante, car une même modification doit être

répercutée autant de fois que le nombre de versions préfixées nécessaires.

C'est pourquoi plusieurs outils en ligne existent afin de pallier ce souci de répétition de l'information. Citons parmi les plus connus http://prefixmycss.com , http://prefixr.com ou encore http://cssprefixer.com.

9.4.1. prefixmycss, prefixr

Ces outils reposent sur un principe identique :

- nous avons construits notre code CSS contenant des règles CSS3 préfixées ou non, en le testant depuis notre navigateur préféré ;
- ce code est parsé par l'outil, lequel nous en renvoi une version préfixée, c'est-à-dire, une version agrémentée de tous les préfixes propriétaires nécessaires pour tous les navigateurs ;
- il nous suffit de remplacer notre code par celui délivré par l'outil et le résultat est instantané.

Fig. 9.10 prefixmycss et prefixr

9.4.2. prefix-free

Fig. 9.11 Logo de prefix-free

Cette librairie développée par Léa Verou (http://leaverou.github.com/prefixfree/) fonctionne différemment. Elle permet de s'affranchir totalement de l'utilisation des préfixes. En effet, votre code CSS doit être dépourvu de préfixes, et ceux-ci sont ajoutés à la volée *via* JavaScript, en fonction du navigateur utilisé. Votre CSS reste donc vierge de préfixe (figure 9.12).

```
box-sizing:border-box;
animation:progress 20s linear infinite;
height:23px;
border-radius:15px;
background:linear-gradient(rgba(255,255,255,.1) 50%,rgba(255,255,255,0) 60%),#09c;
box-shadow:0 1px 0 rgba(255,255,255,.5) inset,0 -1px 0 rgba(0,0,0,.8) inset,0 0 1px black;
}
p{
padding-left:5px;
animation:downloading 2s ease infinite;
}
@keyframes progress{
from{width:10%;}
to  {width:100%;}
}
@keyframes downloading{
from,to{color:#09c;text-shadow:0 -1px 0 rgba(0,0,0,.5);}
50%{color:rgba(0,153,204,.5);text-shadow:0 -1px 0 rgba(0,0,0,.2);
```

Fig. 9.12 Un code CSS sans préfixe

La technique utilisée est la même que celle de Modernizr, à savoir la détection des fonctionnalités dans le navigateur. Cela permets de détecter quelles propriétés CSS sont supportées et lesquelles

sont supportées en versions préfixées.

L'intérêt d'une telle librairie réside dans le fait d'avoir des fichiers CSS plus légers, mais surtout la maintenance (et la compréhension) de ces fichiers s'en trouve facilitée.

Un petit défaut néanmoins. Le navigateur du client doit exécuter JavaScript pour que la librairie puisse fonctionner. De plus, si un problème de réseau survient lors du chargement de la page, les styles CSS3 ne sont pas appliqués. Pour contourner ce problème, une solution orientée serveur (avec un langage tel que PHP ou Node par exemple) aurait pu être envisagée, mais celle-ci pose d'autres soucis, dont le principal est le maintient d'un document faisant état de l'implémentation des propriétés dans les navigateurs (préfixées ou non), car une détection des fonctionnalités n'est plus possible.

9.5. Outils d'aide au débogage

Déboguer du code CSS n'est pas toujours simple, surtout lorsque plusieurs styles s'appliquent à un même élément, *via* le principe de la cascade. Pour nous aider dans cette tâche, les navigateurs offrent aujourd'hui de nombreux outils, appelés généralement outils de développement (figure 9.13). Tous les navigateurs récents en proposent, disponibles depuis les menus, ou depuis le raccourci F12. Ces outils permettent d'accéder à de nombreuses informations, comme l'arbre DOM, la console JavaScript, les requêtes HTTP, les ressources de la page, etc.

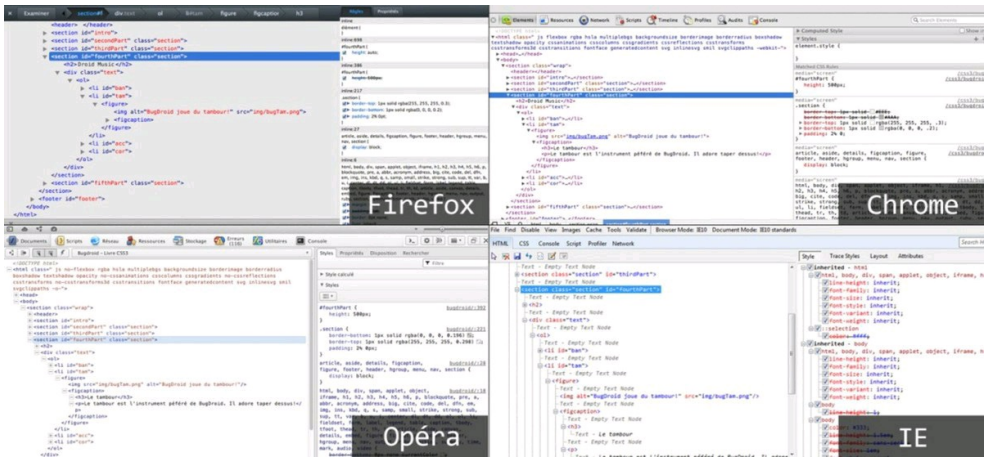


Fig. 9.13 Outils de développement des navigateurs

9.5.1. Visualiser les styles

En ce qui concerne CSS, cette console nous permet de parcourir le DOM en survolant les éléments HTML, et ainsi de visualiser les CSS appliqués pour chaque élément en suivant l'ordre de la cascade. Nous avons également accès aux styles par défaut du navigateur, ainsi qu'à toutes les valeurs des propriétés CSS, telles que la largeur, la hauteur, les marges, etc. Cette fonctionnalité est primordiale pour le débogage.

Il est également possible de désactiver, de masquer ou de modifier le style CSS d'un élément, afin de voir l'effet en « temps-réel ».

De plus, certains de ces outils pour navigateur permettent de faire du *remote debugging*, à savoir du débogage à distance (pages web sur mobile) via un navigateur de bureau. Les dernières versions des principaux navigateurs proposent ces outils adaptés.

9.5.2. Éditer les styles

En lien avec la fonctionnalité d'édition citée précédemment, l'édition de style nous permet d'aller plus loin. En effet, il est possible d'accéder au fichier CSS lui-même, tel qu'il a été chargé par le navigateur, afin par exemple de modifier un ensemble de styles, et de voir instantanément le résultat.

Cette fonctionnalité a été introduite par l'extension Web Developer pour Firefox. Depuis, la version 11 de ce même navigateur inclut ce système, appelé « Éditeur de styles » depuis le menu Développeur Web (figure 9.14).

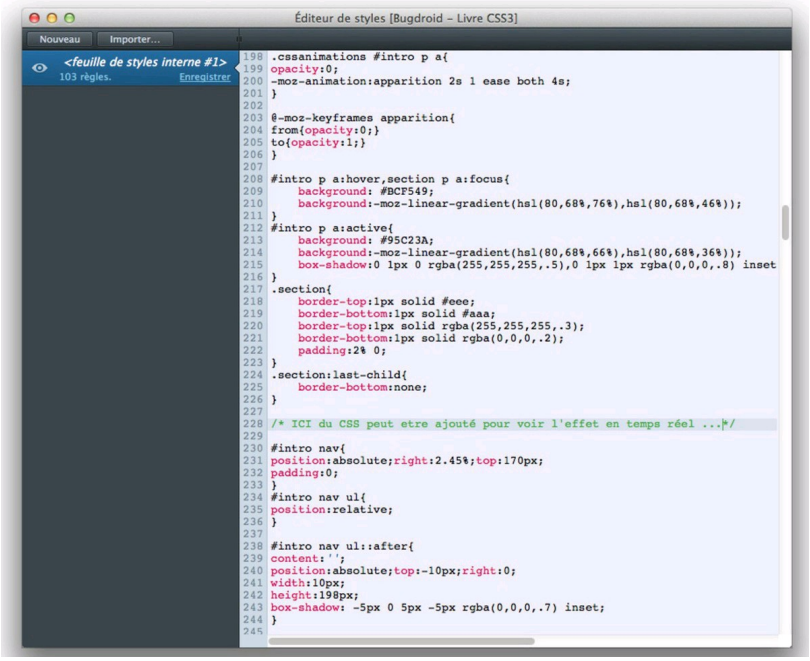


Fig. 9.14 L'éditeur de styles de Firefox

9.5.3. Autres aides

D'autres besoins peuvent s'avérer utile, comme par exemple mesurer quelque chose dans la page, redimensionner le navigateur précisément, récupérer une couleur au sein d'une page et la convertir pour une utilisation depuis d'autres modèles, etc.

Beaucoup d'extensions de navigateurs permettent de réaliser cela (figure 9.15). Citons notamment Web Developer (qui est l'un des plus complets) ou Colorzilla (pipette de couleurs). Vous trouverez de très bons tutoriels qui en détaillent les fonctionnalités.

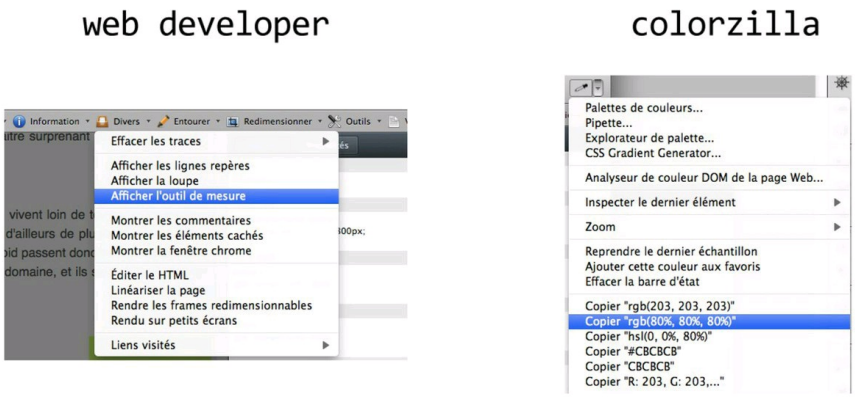


Fig. 9.15 Les extensions Web Developer et Colorzilla

9.5.4. Tester les navigateurs

Le rendu visuel au sein d'un navigateur est également une aide au débogage. Pour cela, il est préférable de disposer de plusieurs versions de chaque navigateur, et si possible, sur des plateformes différentes (Windows XP, Windows 7, Mac OS, Linux, etc.) afin de couvrir le plus large panel possible.

Plusieurs solutions s'offrent à nous pour réaliser cette opération, la plus simple étant la création de machines virtuelles sur un même ordinateur. Chaque machine virtuelle exécute alors son propre environnement et n'interfère pas avec les autres. Il devient donc possible d'y installer une seule version des différents navigateurs.

Par exemple, une machine virtuelle sous Window XP pourrait être pourvue de Firefox 3.6 (Gecko 1.9), d'IE 8 et de Chrome 4, une autre sous Windows 7 pourrait être pourvue de Firefox 4 (Gecko 2) et d'IE 9, et ainsi de suite...

Virtual Box, gratuit, open-source et multi-plateformes, nous aide dans cette tâche.
URL : https://www.virtualbox.org/

9.6. Ressources

9.6.1. Apprentissage de CSS

- **Alsacréations** [FR]
  Communauté francophone dédiée aux standards du web. Très bons articles détaillés de niveau débutant, intermédiaire et avancé notamment sur HTML(5) et CSS(3).
  Forum très actif et pertinent.

  URL : http://www.alsacreations.com

- **CSS Débutant sur Mammouthland** [FR]
  Articles détaillés et efficaces sur l'apprentissage de CSS. Tutoriels de qualité niveau débutant et intermédiaire.

  URL : http://css.mammouthland.net

- **Site du Zéro** [FR]
  Bons tutoriels sur tous les thèmes du web, avec notamment le cours HTML5-CSS3. Accessible aux débutants. Très bien pour acquérir des bases solides.

  URL : http://www.siteduzero.com

- **CSS3Create** [FR]
  Site de démonstration et d'apprentissage de CSS3. De nombreuses explications et tutoriels, niveau intermédiaire et avancé.

URL : http://www.css3create.com

9.6.2. Documentation

- La partie CSS du site officiel du **W3C** [EN]

  URL : http://www.w3.org/TR/CSS

- **Sitepoint Reference CSS** [EN]

  URL : http://reference.sitepoint.com/css

- **Mozilla Developer Network** [EN-FR] (ne concerne pas exclusivement les produits Mozilla)

  URL : https://developer.mozilla.org/fr/CSS/Référence_CSS

- **Safari Developer Reference** [EN]

  URL : http://developer.apple.com/library/safari/#documentation/appleapplications/reference/Safar

- **MSDN Reference** [EN-FR]

  URL : http://msdn.microsoft.com/fr-fr/library/ee836095(v=vs.85).aspx
  URL : http://msdn.microsoft.com/en-us/library/ie/hh673536(v=vs.85).aspx

- **Caniuse** (compatibilté des navigateurs)

  URL : http://caniuse.com

Notes

[1] Solution de repli.

[2] Un fichier .htc contient du code JavaScript. Ce fichier est appliqué sur un élément HTML à l'aide de la propriété CSS behavior qui est reconnue uniquement par Internet Explorer. Le code JavaScript se charge alors de modifier le comportement de cet élément, ou de réaliser diverses opérations.

[3] Le VML (*Vector Markup Language*) est un langage XML ouvert destiné à la création des graphismes vectoriels élaborés en 2D ou 3D (statiques ou animées) sur les pages Web. VML a été soumis comme proposition de standard au W3C en 1998 par Autodesk, Hewlett-Packard, Macromedia, Microsoft et Visio, mais il se trouve en concurrence avec le PGML proposé par Adobe Systems et Sun Microsystems. Après un long examen, le W3C décide de combiner ces deux formats, donnant le jour au format SVG. VML est cependant implémenté dans Internet Explorer (source : http://fr.wikipedia.org/wiki/Vector_Markup_Language).

Aller plus loin !

Tout au long de la lecture de ce livre, nous avons pu remarquer que CSS3 apporte son lot de nouveautés pour tout développeur web. Et ce n'est que le début !

Jusqu'à présent, les implémentations des navigateurs sont surtout axées sur le côté graphique de la norme, avec notamment le support des ombres, des angles arrondis ou encore des dégradés. Cela s'explique par le fait que ce sont les plus simples et les plus rapides à normaliser, et donc à mettre en place. Mais les modules les plus techniques, comme les nouvelles structures de mises en pages ou les transformations 3D, n'en sont encore qu'à leur balbutiement. Ce n'est que lorsque ces spécifications auront atteint le statut de Recommandation et que la totalité des utilisateurs seront équipés de navigateurs compatibles, que l'on pourra mesurer toute la puissance de CSS3.

Mais pour cela, il faut continuer dans la même direction, et c'est pourquoi le W3C poursuit sans relâche son travail de normalisation et CSS4 pointe même déjà le bout de son nez avec de nouvelles spécifications publiées. Ce travail est nécessaire pour un web en constante évolution.

Terminons ce livre en vous mettant un peu l'eau à la bouche : nous allons vous présenter quelques brouillons de spécifications de CSS, qui nous paraissent très prometteurs pour le futur du design web.

10.1. Effets graphiques

Même si CSS3 nous a déjà apporté beaucoup en termes d'effets graphiques, de nombreux effets nous sont encore impossible. La spécification tente alors de répondre à de nouveaux besoins en proposant par exemple une fonction de transition des images d'arrière-plan, ou encore une fonction qui convertit un élément de la page en image, afin par exemple de réaliser des arrière-plans animés.

De plus, pour compléter notre gamme d'effets, et pour se rapprocher encore de la puissance des logiciels graphiques comme Photoshop, la spécification nous propose des filtres, qui nous permettent de régler le contraste, la luminosité, le flou ou la netteté de n'importe quel élément d'une page web.

10.1.1. SVG

Dans le chapitre sur les effets graphiques, nous avons décrit la façon dont les graphiques SVG peuvent êtres combinés avec un document HTML, que ce soit depuis la balise `img`, ou en CSS depuis la propriété `background-image`.

Cependant, le lien entre CSS et SVG est encore plus étroit que cela, puisqu'il est possible de styler un élément SVG directement depuis CSS. Et comme SVG peut être inclus au sein du HTML, la même feuille de styles peut être utilisée, et ce, sans préciser l'espace de nom !

La norme SVG 1.1 définit les propriétés CSS applicables aux éléments SVG. Pour faire simple, seules les propriétés de polices, de textes et d'affichage de CSS2.1 sont utilisables. Cependant, la norme fait état d'autres propriétés, définies uniquement au sein de SVG. Ces propriétés sont surtout liées à l'apparence. Citons par exemple les propriétés de remplissage `fill-*`, ou les propriétés de

contours : stroke-*.

Grâce à l'association de CSS et de SVG, il est donc possible de faire du dessin vectoriel tout en gardant la sémantique de nos éléments.

HTML

```
<!DOCTYPE html>
<html>
	<body>
		<h1>Dessin vectoriel en SVG</h1>
		<svg>
			<rect x="50" y="50" width="200" height="200" />
			<polygon id="poly" points="200,50 300,200 100,200" />
		</svg>
</body>
</html>
```

CSS

```
h1{
	color: navy;
}
rect{
	fill: red;
	stroke-fill: black;
	stroke-width: 10px;
}
#poly{
	fill: lime;
	stroke-fill: green;
	stroke-width: 10px;
	stroke-linejoin: miter;
	stroke-dasharray: 15, 10;
}
```

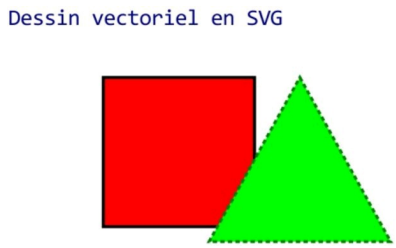


Fig. 10.1 Rendu dans notre page HTML

Nous avons donc un élément svg qui contient un rect et un polygon. Ces deux formes sont dimensionnées en SVG, mais sont stylées en CSS (figure 10.1) .

De plus, le W3C propose désormais une spécification commune pour les transformations CSS et SVG. Et les transitions fonctionnent également ! La syntaxe CSS est donc applicable sur des éléments SVG (figure 10.2).

```
rect{
	...
	transition: transform 2s ease;
}
rect:hover{
```

```
	transform: rotate(45deg);
}
```

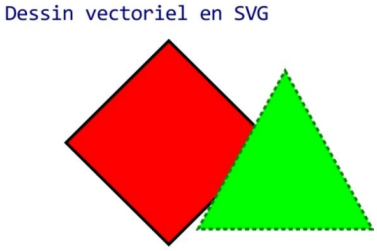


Fig. 10.2 Transformation CSS sur un élément SVG

Les dernières versions de Chrome supportent déjà les transitions et transformations sur des éléments SVG. Nul doute que les autres navigateurs vont très rapidement suivre.

10.1.2. cross-fade

La fonction cross-fade() permet de combiner deux images en spécificiant la valeur de transition entre elles par un pourcentage. Cela permet par exemple de jouer sur la transparence d'images appliquées en arrière-plan (figures 10.3 et 10.4), ou encore d'animer une transition entre deux images.

```
div{
	background-image: cross-fade(url('image1.png'), url('image2.png'), 30%);
}
```

Fig. 10.3 Transparence entre deux images

```
div:hover{
	background-image: cross-fade(url('image1.png'), url('image2.png'), 80%);
}
```

Fig. 10.4 Transparence différente au survol

Cette fonction, présente dans le brouillon du *CSS Image Values and Replaced Content Module de niveau 3* en juillet 2011, a disparu dans la nouvelle version de septembre 2011. Elle a été déplacée

pour une étude plus poussée dans CSS4.

Une version de test a pourtant été implémentée au sein de Webkit et est utilisable depuis Chrome 17. Cette fonction doit être préfixée de telle manière -webkit-cross-fade().

10.1.3. image

La fonction image() devrait progressivement remplacer la fonction url() dans le cas du chargement d'images, puisqu'elle permet d'aller beaucoup plus loin, en ajoutant par exemple les fonctionnalités suivantes :

- Découpe de portion d'images
- Ajout d'images alternatives
- Conversion d'une couleur en image
- Définition du sens de l'image

Découpe de portion d'images

Lors de l'appel d'une image, des paramètres de positions et de tailles peuvent êtres ajoutés afin de créer une nouvelle image, à partir d'une portion de l'image source. La gestion des *sprites* s'en trouve alors simplifier. La syntaxe est la suivante :

```
div{
	background-image: image('sprite.png#xywh=0,0,20,20');
}
```

Dans ce cas, l'image *sprite.png* est découpée depuis la position 0,0 sur une taille de 20,20, et appliquée en arrière-plan de notre élément.

Ajout d'images alternatives

Dans le cas où l'image utilisée ne serait pas reconnue par le navigateur (problème de format par exemple), une deuxième image peut être spécifiée. De cette façon, une seule propriété CSS doit être écrite :

```
div{
	background-image: image('image.svg', 'image.png', 'image.gif');
}
```

Ici, l'image au format SVG est d'abord ajoutée. Les navigateurs qui ne connaissent pas ce format vont télécharger le PNG, et ainsi de suite.

Conversion de couleur

Avec image(), une couleur peut être utilisée. Celle-ci sera convertie en image pour une utilisation depuis une propriété qui n'accepte pas une valeur de type color.

```
div{
	background-image: image(red);
}
```

Définition du sens de l'image

À l'instar de la définition du sens d'écriture avec la propriété direction, la fonction image() permet de

définir le sens d'une image. Par exemple, une image de flèche appliquée comme puce dans une liste pourrait être affichée inversée, si l'écriture de la page est inversée.

```
ul li{
	list-style-image: image('fleche.png' ltr);
}
```

Dans cet exemple, l'image *fleche.png* sera affichée à l'endroit si le texte est écrit de la gauche vers la droite, et sera inversée si le texte est écrit de la droite vers la gauche. Cela revient à faire une transformation scaleX(-1).

Note : la fonction image() n'est actuellement supportée par aucun navigateur.

10.1.4. element

La fonction element() initialement présente au sein du module *Image Level 3*, convertit n'importe quel élément d'une page web en image. Cette image, qui est « temps-réel » peut ensuite être utilisée en arrière-plan d'un autre élément *via* background-image.

Avec cette fonction, nous pouvons par exemple créer un diaporama tout en ajoutant un reflet dynamique sur le bas (figure 10.5).

HTML

```
<div id="diapo"><img src="image" alt="" /></div>
<div id="reflet"></div>
```

CSS

```
#reflet{
	background: element(#diapo);
	transform: scaleY(-1);
}
#reflet::after{
	background: linear-gradient(white, rgba(255,255,255,0));
}
```

Fig. 10.5 Le reflet est dynamique et créé avec element()

Grâce à element(), il devient aussi possible d'utiliser le rendu de la balise canvas ou video, tel une image dynamique. Il est donc possible de créer un arrière-plan animé.

La spécification fait également état de l'utilisation de JavaScript pour générer des éléments et les stocker dans des objets, que nous pouvons ensuite appeler depuis la fonction element(). Ainsi, le rendu

d'un canvas peut être accessible sans pour autant être présent sur la page.

Une version de test de cette fonction est actuellement supportée par Firefox, en ajoutant le préfixe propriétaire, de cette façon : -moz-element(). Néanmoins, son étude est repoussée dans CSS4.

10.1.5. Les filtres CSS

Les filtres CSS sont une révolution pour les designers web. Ils permettent de s'affranchir des traitements lourds que nous appliquons à nos images depuis nos logiciels graphiques. De la même façon que l'apparition des ombres et des dégradés ont révolutionné le design web, les filtres CSS vont nous permettre de créer de nouveaux effets visuels en nous offrant un gain de temps énorme dans la maintenabilité de nos images.

L'intérêt de disposer de tels filtres est immense. Nous pourrions envisager une galerie photo où les éléments non survolés deviennent flous pour créer un effet de profondeur. Ou encore une image en niveaux de gris qui se colore au survol. Ou même les deux (figure 10.6) !

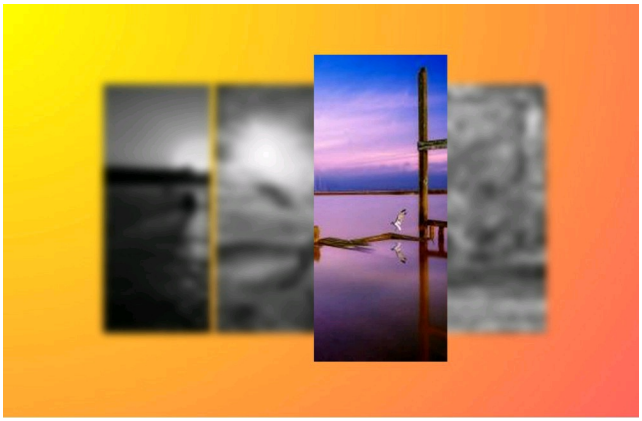


Fig. 10.6 Utilisation des filtres CSS

Actuellement, pour obtenir ce rendu, il n'existe pas de solution simple. Généralement, un tel résultat passe par l'utilisation de la balise canvas et de beaucoup de code JavaScript.

Certains développeurs ont alors eu l'idée d'utiliser la puissance des filtres SVG, qui offrent justement ces possibilités, afin de les appliquer sur du contenu HTML. Cependant, bien que le support de SVG soit en nette progression, la balise foreignObject doit être utilisée. Cette balise permet d'ajouter du contenu HTML dans le balisage SVG, afin d'utiliser des fameux filtres sur du HTML. Malheureusement, cette technique n'est pas très pratique, et très peu supportée.

Firefox a alors ouvert la possibilité d'utiliser les filtres SVG directement sur du contenu HTML, et ce depuis la version 3.5. Pour cela, un fichier SVG externe doit être utilisé, lequel contient le balisage nécessaire à la création des filtres. Ce fichier est ensuite appelé depuis CSS via la propriété filter, comme ceci :

HTML

```
<div>Un texte</div>
```

CSS

```
div{
	filter: url('filtres.svg#flou');
}
```

Dans cet exemple, le fichier *filtres.svg* est chargé, et le filtre dont l'identifiant est `#flou` est appliqué sur l'élément. Toute la puissance des filtres SVG est alors applicable à n'importe quel élément HTML.

Fort de cette expérience, une spécification a vu le jour (encore en brouillon éditeur) afin de standardiser l'utilisation de ces filtres SVG, le *Filter Effect 1.0*.

> **URL** : https://dvcs.w3.org/hg/FXTF/raw-file/tip/filters/index.html.

Cette spécification propose trois concepts pour l'utilisation de ces filtres :

- la syntaxe raccourcie ;
- les filtres SVG « classiques » ;
- les *shaders*.

Les raccourcis

Cette partie de la spécification est certainement celle qui sera le plus utilisée par les développeurs web. En effet, nul besoin d'apprendre la syntaxe des filtres SVG, puisque le brouillon définit dix mot-clés, qui sont les raccourcis de filtres SVG directement codés au sein du navigateur. Les filtres disponibles sont les suivants :

- *grayscale*
- *sepia*
- *saturate*
- *hue-rotate*
- *invert*
- *opacity*
- *brightness*
- *contrast*
- *blur*
- *drop-shadow*

Ainsi, l'utilisation de ces filtres courants s'en trouve facilitée. Par exemple, la syntaxe est la suivante pour un effet de flou de 5 px (figure 10.7).

Application d'un flou

```
img:nth-child(3){
	filter: blur(5px);
}
```

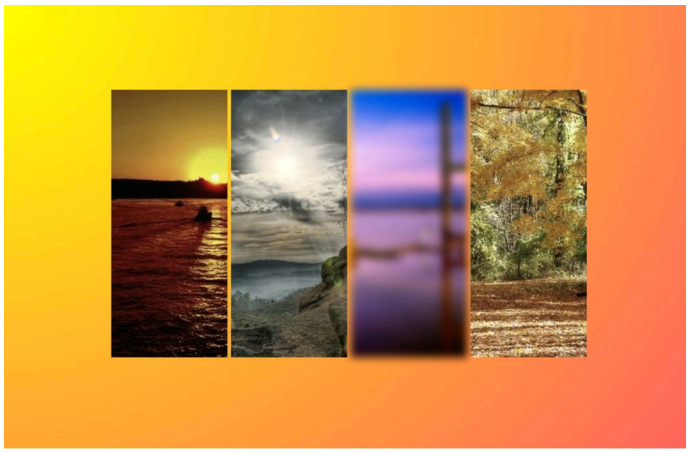


Fig. 10.7 Le troisième élément est flouté

Et voici un exemple d'utilisation de deux filtres, niveaux de gris et contraste (figure 10.8) :

```
img:nth-child(3){
	filter: grayscale(1) contrast(2);
}
```

Fig. 10.8 Le troisième élément est en niveaux de gris avec le contraste accentué

Cette notation raccourcie induit un énorme autre avantage: la possibilité d'animer facilement les propriétés de filtres, grâce aux transitions et animations CSS. Ainsi, ce code permet de passer d'un flou de 0 à un flou de 5 px sur un paragraphe (figure 10.9) :

```
<p>HELLO!</p>
p{
	filter: blur(0);
	transition: filter 2s ease;
}
p:hover{
	filter: blur(5px);
}
```

Fig. 10.9 Progression de l'animation du flou

La notation des filtres SVG avec raccourcis est supportée depuis Chrome 19 en version préfixée. La notation pour un effet de flou est donc : -webkit-filter: blur(5px);.

Les filtres « classiques »

Les filtres dits « classiques » sont définis en SVG au sein d'un fichier externe. L'utilisation de ces filtres permet des traitements beaucoup plus poussés que les simples raccourcis. Cela revient exactement à ce que Firefox permet depuis longtemps.

Nous n'allons pas rentrer dans le détail mais un filtre SVG est une succession d'opérations graphiques définies par des primitives de filtre. Chaque primitive effectue donc une seule opération, qui est appliquée sur un élément source (déjà modifié par d'autres primitives ou non), et qui produit une sortie sous forme d'image. L'ensemble des primitives produit alors un filtre complexe.

Les primitives de filtres sont multiples et définies au sein de la spécification SVG1.1. Voici un aperçu qui provient du site du W3C :

```
<filter id="MyFilter" filterUnits="userSpaceOnUse" x="0" y="0" width="200" height="120">
	<feGaussianBlur in="SourceAlpha" stdDeviation="4" result="blur"/>
	<feOffset in="blur" dx="4" dy="4" result="offsetBlur"/>
	<feSpecularLighting in="blur" surfaceScale="5" specularConstant=".75"
	specularExponent="20" lighting-color="#bbbbbb" result="specOut">
		<fePointLight x="-5000" y="-10000" z="20000"/>
	</feSpecularLighting>
	<feComposite in="specOut" in2="SourceAlpha" operator="in" result="specOut"/>
	<feComposite in="SourceGraphic" in2="specOut" operator="arithmetic" k1="0" k2="1" k3="1" k4="0" result="litPai
	<feMerge>
		<feMergeNode in="offsetBlur"/>
		<feMergeNode in="litPaint"/>
	</feMerge>
</filter>
```

Ici, on retrouve plusieurs primitives : un flou gaussien (<feGaussianBlur>), un décalage (<feOffset>), un effet de lumière (<feSpecularLighting>), une combinaison d'images (<feComposite>) et une fusion des couches (<feMerge>).

Une bonne maitrise des filtres SVG permet d'obtenir des effets saisissants (figure 10.10).

Fig. 10.10 Un filtre SVG appliqué sur un élément HTML

Les shaders

Les *shaders* sont un moyen d'aller encore plus loin avec les filtres. En effet, les *shaders* sont des programmes qui modifient la géométrie des éléments en suivant un maillage (les *vertex shaders*) et en modifiant la couleur de chaque pixel (les *fragments shaders* ou *pixel shaders*) afin de recréer un effet de 3D par exemple.

Grâce aux *shaders*, il est par exemple possible de créer un effet de drapeau, ou de vague, sur n'importe quel élément HTML ou SVG, et d'appliquer une modification des couleurs calculées en fonction de différents paramètres. La déformation est réalisée avec un *vertex shader* (fichier .vs) et la modification des couleurs avec un *fragment shader* (fichier .fs).

La fonction custom() doit être utilisée depuis la propriété filter, comme ceci :

Application d'un filtre complexe

```
div{
	filter: custom( url('vague.vs'), url('reflet.fs'), 40 40, nombre 10, hauteur 10);
}
```

Dans cet exemple, le fichier vague.vs se charge de déformer l'élément, le fichier reflet.fs traite les couleurs, 40 40 crée le maillage, *nombre* et *hauteur* sont deux variables qui peuvent être utilisés par les *shaders*.

La spécification définit également la possibilité d'animer ces *shaders via* les transitions et animations CSS. Notre exemple précédent peut donc être modifié ainsi :

```
div{
	filter: custom( url('vague.vs'), url('reflet.fs'), 40 40, nombre 10, hauteur 10);
	transition: filter 5s ease;
}
div:hover{
	filter: custom( url('vague.vs'), url('reflet.fs'), 40 40, nombre 10, hauteur 50);
}
```

Lors du survol avec la souris, le paramètre hauteur est modifié. Le fichier vague.vs qui l'utilise va alors déformer l'élément, et une transition entre les deux états va être effectuée.

Note : les shaders reposent sur le même langage que WebGL, à savoir: OpenGL ES Shading Language

10.2. Positionnement très avancé

En termes de mise en pages, CSS3 introduit de nouveaux systèmes. Au cours de cet ouvrage, nous avons déjà pu détailler le *Flexible Box Layout*, le *Grid Layout* ainsi que le *Multi-column Layout*. Cependant, la spécification fait état d'autres systèmes de positionnement qui viennent s'ajouter aux trois cités précédemment. Nous allons donc nous intéresser ici à deux d'entre eux, il s'agit des modules *Exclusions and Shapes Module* et *Regions Module*. Ces deux modules sont passés à l'état de *Working Draft* depuis fin 2011.

10.2.1. Exclusions

Le module *Exclusions and Shapes,* dont le premier *Working Draft* date du 13 décembre 2011, repose sur deux concepts : les exclusions et les formes.

Les exclusions permettent de spécifier un élément qui sera contourné par le flux de la page. Contrairement à float, tous les éléments de la page peuvent être contournés, et cela peu importe leur mode de positionnement et leur position dans la page.

Les formes, basées sur SVG, sont des objets géométriques définis pour un élément. La combinaison des deux notions permet de créer des mises en pages complexes où le contenu s'affiche en contournant une forme géométrique telle un cercle ou un polygone complexe (figure 10.11).

Fig. 10.11 Cas pratique d'utilisation des Exclusions CSS

wrap-flow

La propriété wrap-flow définit un élément comme étant une exclusion. Les valeurs possibles sont auto, both, start, end, maximum et clear. La valeur par défaut est auto, ce qui signifie qu'aucune exclusion n'est créée (sauf pour les éléments en float).

Lorsqu'un élément est une exclusion, les textes contournent cet élément sans jamais le superposer (figure 10.12).

HTML

```
<div id="exclusion">Cet élément est une exclusion CSS</div>
<div>
    <p>...</p>
</div>
```

CSS

```
#exclusion{
    position: absolute;
}
```

Fig. 10.12 Les différentes applications de wrap-flow:

Dans la figure 10.12, notre élément exclusion est en position: absolute, mais le principe fonctionne également avec tous les modes de positionnement, comme le *Flexible Box Layout* ou le *Grid Layout*.

wrap-margin, wrap-padding

Ces deux propriétés permettent de modifier les marges internes et externes d'une exclusion CSS. Le fonctionnement est identique aux propriétés classiques margin et padding.

shape-inside, shape-outside

Les deux propriétés shape-inside et shape-outside permettent de créer un objet géométrique pour un élément, afin que le flux le contourne en suivant cette forme. Les formes sont créées *via* des fonctions qui simulent les balises SVG classiques.

Voici un exemple ou le texte suit un cercle, grâce à la fonction circle() qui s'utilise de cette manière : circle(centre_x, centre_y, rayon).

HTML

```
<div id="exclusion"></div>
<div>
    <p>...</p>
</div>
```

CSS

```
#exclusion{
    position: absolute;
    wrap-flow: both;
    shape-outside: circle(50%, 50%, 50%);
    border-radius: 50%;
}
```

```
shape-outside:
circle( 50%, 50%, 50%);
```

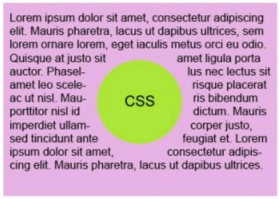


Fig. 10.13 Le texte suit la forme

Dans la figure 10.13, le flux de la page suit la forme définie avec la fonction circle(). Le fait que l'exclusion soit un cercle est simplement dû à l'utilisation de border-radius. L'élément HTML est toujours un rectangle.

L'inverse est possible, c'est-à-dire à l'intérieur d'une forme, avec l'utilisation de shape-inside.

La spécification précise également la possibilité d'utiliser la couche alpha d'une image comme forme géométrique d'exclusion. Cet aspect doit encore être étudié très précisément.

Compatibilité

Les exclusions sont très expérimentales et ne sont actuellement supportées par aucun navigateur. Cependant, la première version « preview » d'IE10 supportait la règle float: positioned (qui fait partie d'une ancienne spécification à la base des exclusions CSS). La dernière version d'IE10, toujours en « preview », supporte, quant à elle, une petite partie de la spécification actuelle. Les exclusions sont présentes dans les *Nightly Build* de Webkit, mais elles ne sont pas encore activées par défaut.

10.2.2. Régions

Le module *Regions Module* définit un système de positionnement où le flux de la page est affiché de manière continue au sein de plusieurs blocs distincts. Le contenu d'un élément HTML est donc réparti sur plusieurs emplacements de la page, que l'on appelle les régions.

Ces régions peuvent être positionnées dans la page, en se basant sur n'importe quel système de mise en pages (figure 10.14).

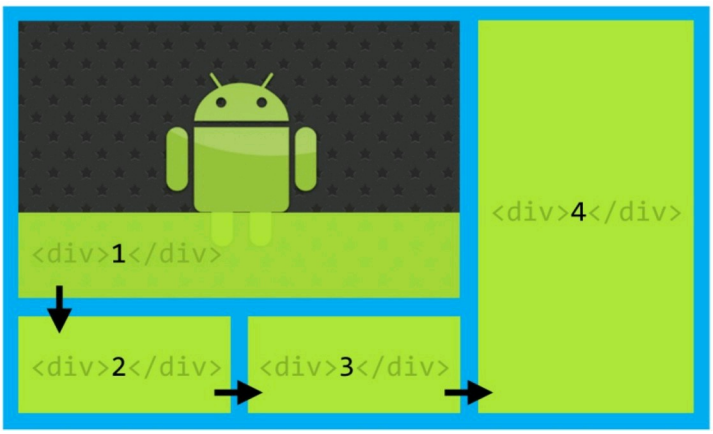


Fig. 10.14 Positionnement des régions

Les régions sont basées sur un concept appelé *named flow*, que l'on pourrait traduire par *contenu nommé*. Le contenu d'un élément HTML doit alors être ajouté à un *contenu nommé*, puis il est récupéré pour être affiché au fil de plusieurs autres éléments.

flow-into

La propriété flow-into permet de spécifier le *contenu nommé* d'un flux. Par exemple, un contenu textuel pourrait être nommé *texte*, comme ceci :

HTML

```
<article>
	<h1>Le titre de la page</h1>
	<p>Le premier paragraphe...</p>
	<p>Le second paragraphe...</p>
	<p>Le troisième paragraphe...</p>
	<h2>Sous-titre</h2>
	<p>La dernière sous-partie...</p>
</article>
```

CSS

```
article{
	flow-into: 'texte';
}
```

Dans ce cas, tout le contenu HTML de la balise article est nommé *texte*. Ce contenu peut maintenant être affiché ailleurs.

flow-from

La propriété flow-from affiche le *contenu nommé* défini avec flow-into, au sein des éléments ciblés. Ainsi modifions notre exemple, en ajoutant quatre éléments HTML :

HTML

```
<article>...</article>
<div class="regions"></div>
<div class="regions"></div>
<div class="regions"></div>
<div class="regions"></div>
```

Le contenu nommé est affiché dans les éléments .regions :

CSS

```
div.regions{
	flow-from: 'texte';
}
```

Le flux de la balise article est alors affiché au sein de la première div.regions et continue au sein de la seconde, de la troisième et de la dernière, et cela peut importe leurs positions dans la page, et leurs modes de positionnement. En clair, dès que la première région est « remplie », le contenu est alors stoppé, et la suite est affichée dans la seconde région, et ainsi de suite (figure 10.15).

```
article{
  flow-into: 'texte';
}

div.regions{
  flow-from: 'texte';
}
```

Fig. 10.15 Le contenu est automatiquement réparti

L'intérêt d'une telle structure réside dans le fait que le contenu peut être affiché sans se soucier de la position des éléments. Cela permet, entre autre, de positionner les éléments différemment en fonction des résolutions ou des tailles d'écrans, tout en conservant le flux de la page affiché de régions en régions.

Contrôles

Une telle structure impose bien entendu de multiples contraintes, auxquelles la spécification tente de répondre. Citons en particulier les propriétés break-before, break-after et break-inside, qui permettent de forcer un saut de régions avant, après ou au sein d'un élément. Citons également la propriété region-overflow qui gère le cas ou le contenu déborde de la dernière région possible pour le flux.

Terminons par la règle @region qui permet, à la manière des *medias-queries*, de cibler les éléments actuellement au sein d'une région particulière. Ainsi, il devient possible de les styler différemment, comme ceci :

```
@region div.regions:nth-child(2){
	p{
		background: gray;
	}
}
```

Cette règle ajoute un arrière-plan gris, à tous les paragraphes, actuellement dans la deuxième région.

Compatibilité

Les régions CSS sont également très expérimentales, actuellement supportées au sein d'IE10 et depuis Chrome 15. Cependant, les syntaxes diffèrent légèrement de la spécification actuelle.

Index

accélération, 233, 237
accessibilité, 4, 140, 229
amélioration progressive, 34, 236, 259, 274, 275
Android, 31, 218
animations, 238, 273
arrière-plans multiples, 99
attr(), 80
background-attachment, 101
background-clip, 105
background-origin, 106
background-position, 102
background-repeat, 100
background-size, 107
block, 182
border-collapse, 202
border-image, 114
border-radius, 97
border-spacing, 201
box-shadow, 92
box-sizing, 191
butineur, 19
calc(), 79
Candidate Recommandation, 12
césure, 172
Chrome, 25
circle, 122
clear, 177
closest-corner, 125
closest-side, 124
coins arrondis, 96
color-stop, 130
commentaires conditionnels, 36
compteurs, 61
contain, 108, 128

couleurs, 63
counters, 62
cover, 109, 129
crénage (kerning), 164
cross-fade(), 294
CSS, 6, 39
CSS3
    modules, 17
    PIE, 277
    sélecteurs, 42
    standard, 8
    utilisation, 32
cubic-bezier, 234
currentColor, 65
curseurs, 81
débogage, 291
dégradation gracieuse, 33, 95, 98, 278
dégradés, 118
détection
    fonctionnalités, 38, 284
    navigateurs, 37
device-width, 215, 223
display, 192
échelle, 256, 267
Editor's Draft, 9
element(), 296
ellipse, 123
em, 72, 137
EOT, 146, 148
erreur 404, 149
évènements JS, 242
exclusions, 300
fallback, 120, 279
farthest-corner, 127
farthest-side, 126
faux-gras, 151
faux-italique, 152
filtres CSS, 297

Firefox, 23
Flexible Box Layout (flexbox), 206
float, 52, 176, 301
flou, 91, 93, 298
font, 139
font-family, 132
font-feature-settings, 161
font-size, 134
font-size-adjust, 157
font-stretch, 156
font-variant, 159
Fontspring, 150
Fontsquirrel, 155
formulaires, 82
fr, 74, 208
frameworks, 230, 244
Gecko, 24
GIF, 111
glyphes, 162
Google web fonts, 153
Grid Layout, 207
hauteur d'x, 158
height, 186
hsl, 69
hsla, 70, 87
HTC, 280
hyphens, 171
image, 295
inclinaison, 253, 265
initial-scale, 217
inline, 183
inline-block, 193
inset, 94
Internet Explorer, 21
iPhone, 30, 214
JavaScript, 243, 283, 290
JPG, 112
jQuery, 245

Last Call, 11
layout, 181
ligatures, 163
linear-gradient(), 119
listes, 59
local(), 154
margin, 187
matrice, 257, 269
matrix, 258, 268
Media-queries, 221
mobile, 212
modèle de boîtes, 179
Modernizr, 131, 276, 282
modules CSS3, 16
moteur de rendu, 20
Multi-column Layout, 205, 227
navigateur, 18
Netscape Navigator, 5
ombres, 89
opacity, 66, 85
OpenType, 160
Opéra, 28
orientation, 224, 226
OTF, 145
overflow, 190
overflow-wrap, 169
padding, 188
perspective, 271
PNG, 113
polices
    formats, 142
    génériques, 133
positionnement, 173, 180
    absolu, 178
    avancé, 204
    flottant, 175
    tableaux, 174
prefix-free, 289

préfixes, 15, 35, 286
prefixmycss, 287
prefixr, 288
preserve-3d, 270, 272
Presto, 29
Proposed Recommandation, 13
pseudo-classe, 43, 47
pseudo-éléments, 51
Quirks (mode), 189
radial-gradient(), 121
Recommandation, 14
régions, 302
rem, 73, 138
resize, 83
resolution, 225, 228
Responsive design, 211
ressources, 292
rgb, 67
rgba, 68, 86
rotate, 251, 263
rotation, 250, 264
round, 104, 116
Safari, 27
scale, 255, 266
screen-width, 216
sélecteur, 41
Selectivizr, 285
séparation fond-forme, 3
shaders, 299
skew, 254
smartphones, 210
space, 103, 117
standards web, 1, 7
steps, 235, 241
stretch, 115
SVG, 110, 143
table, 184, 196
table-caption, 203

table-cell, 198
table-layout, 199
table-row, 197
text-overflow, 168
text-shadow, 90
text-space-collapse, 165
text-wrap, 167
transform, 247
transformations
    2D, 246
    3D, 260
transitions, 231
translate, 249, 261
translation, 248, 262
transparence, 84
transparent, 64, 88
Trident, 22
TTF, 144
unités
    absolues, 76, 135
    angles, 77, 252
    relatives, 71, 136
    temps, 78, 232, 240
vertical-align, 194, 200
viewport, 213
vw, vh, vmin, 75
W3C, 2, 293
Webkit, 26
white-space, 166, 195
width, 185
WOFF, 147
word-wrap, 170
Working Draft, 10
z-index, 209, 281
zoom, 220